UNIVERSITA’ DEGLI STUDI DI BARI ALDO MORO

Dipartimento di Informatica

CORSO DI LAUREA TRIENNALE
IN INFORMATICA

TESI DI LAUREA IN
SVILUPPO DEI VIDEOGIOCHI

TITOLO DELLA TESI

A Natural Language Processing Tool for Supporting the Development of Videogames

Relatori
Prof. Pierpaolo Basile
Prof.ssa Berardina De Carolis

Laureando
Giuseppe Volpe

Anno Accademico 2022/2023

Acknowledgements

I would like to offer my sincere thanks firstly to my relator Pierpaolo Basile, who has accompanied me through my university career with his courses, in which he amplified my interest towards this field, that was already intense, with his passion and his knowledge.

A special thanks goes to my family, especially to my mother, my father, and my sister, which always supported me during my studies, personally and economically, without them it would have been incredibly harder to achieve this result.

Another important credit goes to those who helped me with some aspects of this project that I was struggling with, like my uncle Alessandro, who helped me setting up the single page application which has been essential for this project, or my friend Riccardo, who helped me with aspects related to game design and graphics.

Lastly, I want to thank my friends, Pietro, Rossella and the others, that helped me to keep in contact with the social aspects of my life when in the days in which studying was taking over my time.

Index

ABSTRACT

NLP, or **Natural Language Processing**, is a field of artificial intelligence focused on enabling computers to several tasks like understanding, interpreting, and generating natural language, which is the language used by humans. It includes techniques like text analysis, sentiment analysis, machine translation, entities recognition, part of speech tagging and more. NLP applications are wide-ranging, including voice assistants, language translation services, content recommendation systems, and sentiment analysis for social media. Deep learning models, such as recurrent neural networks (RNNs) and transformers, have significantly advanced NLP by improving language understanding and generating more coherent responses. NLP continues to evolve, empowering machines to comprehend and communicate with humans more effectively.

Videogames were born in the mid-twentieth century, as electronic entertainment. One of the cornerstones of this new type of entertainment, "Tennis for Two", was created in 1958 by William Higinbotham. A lot of time has passed, and the field has constantly evolved, becoming one of the most profitable fields of entertainment, but we only had few poor interactions between videogames and Artificial Intelligence technologies like Natural Language Processing.

The aim of this work is to show how this objective could be easily achieved by developing a simple **tool** which, integrated with standard game development practices, can be used **to apply NLP to videogames**.

KEYWORDS

Natural Language Processing, Artificial Intelligence, Videogames, Non-Playable Characters, Deep Learning

CHAPTER 1 – Introduction

Recent advances achieved in the field of Natural Language Processing (NLP), the branch of artificial intelligence that allows machines to understand the ambiguities of human language, are outstanding.

Let's think about the introduction of the GPT models, **Large Language Models (LLM)** which can handle a conversation unbelievably fluidly and plausibly with the end-user, solving various NLP tasks, like Question Answering, Machine Translation, Code Generation, and others.

Although NLP models have become so powerful, some fields of technology by now haven't benefited from these advances in a proper way. One of these fields is the one of videogames, which I decided to deal with for my long-standing passion.

What I want to do with this project is to make a tool which allows the developers who have no knowledge about the mechanisms behind Artificial Intelligence to create **Non-Playable Characters (NPCs)** that can understand the player's sentences without resorting to the use of rigid pre-programmed commands based on key-words, which can be really time consuming for the developers, and could limit the freedom of the gamers.

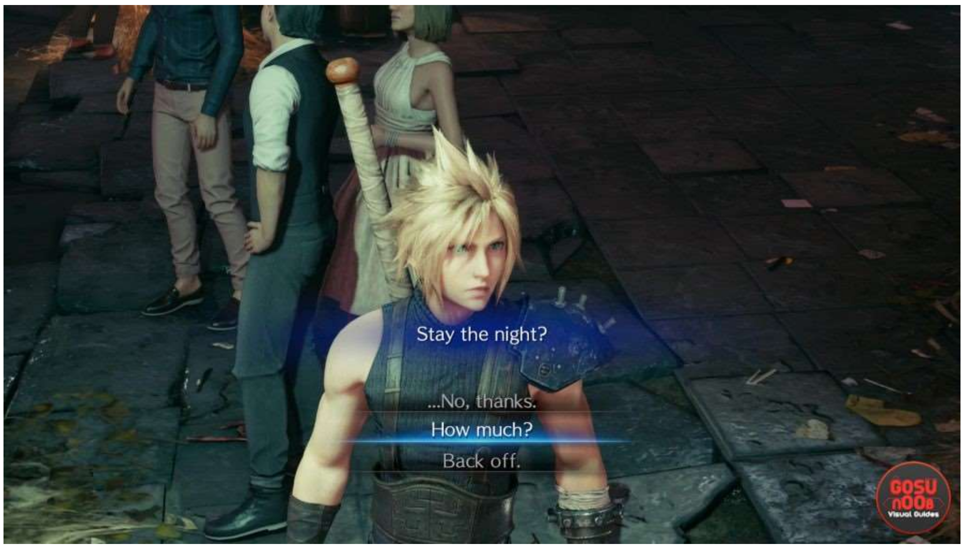


Figure 1 - Pre-programmed dialogues [https://www.gosunoob.com/final-fantasy-vii-remake/stay-the-night-no-thanks-how-much-back-off/]

In the book “Introduction to Game Development”, written by Steve Rabin, the videogame is described as an **artifact**, a thing made with an intended function that someone has made to be used. The aim of this kind of artifacts is to play, just for fun, without other reasons. But how does a game work? It imposes **rules** in the act of playing.

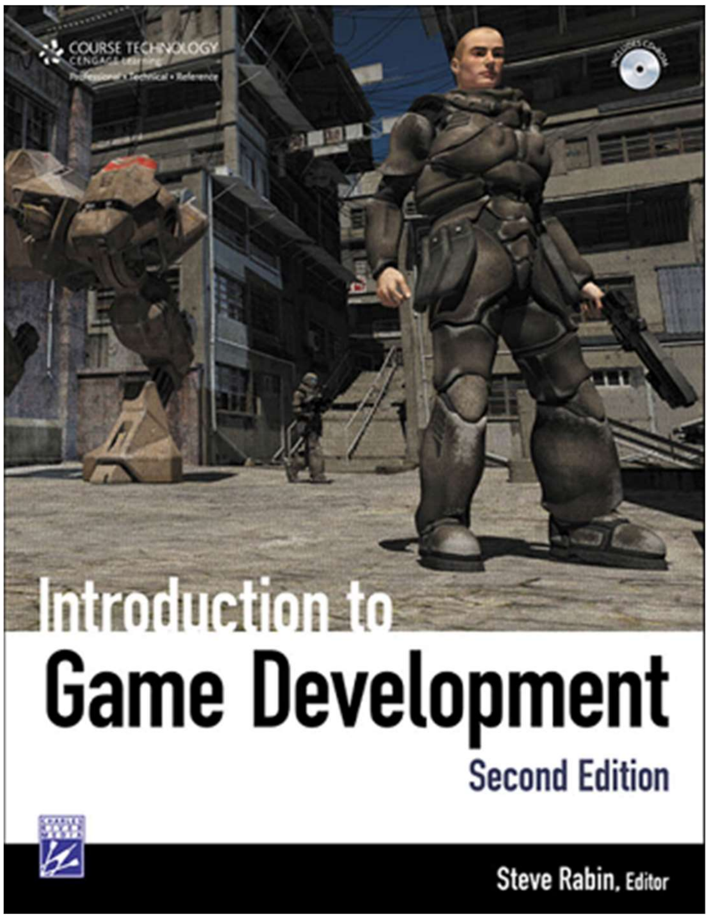


Figure 2 - Steve Rabin's book

Here lies one of the most significant problems. Videogames, in fact, even though it could sound counter intuitive, are artifacts composed of rules which guarantee the structural integrity of the product. How can you bring Artificial Intelligence into videogames without breaking the rules that characterize the game world? This question will be further explained.

In chapter 2, the focus will be directed above the state of the art of NLP research in this field, which will highlight how this problem, at the moment, has not been dealt in an exhaustive way.

CHAPTER 2 – State of art

2.1- Videogames as data source

A very frequent approach to link NLP with videogames is to use the huge source of textual contents related with gaming products to build datasets designed to train Language Models.

2.1.1 – "Natural Language Processing in Game Studies Research: An Overview"

In the research "Natural Language Processing in Game Studies Research: An Overview" [José P. Zagal, Noriko Tomuro, Andriy Shepitsen], they talk about how Natural Language Processing, applied to videogames reviews could give a huge contribution to the research field of "Game Studies".

Game studies research is an interdisciplinary field that examines various aspects of games, analysing them from several point of views: cultural, social, psychological, and aesthetic. It deals with a wide range of topics, including game design, player behaviour, narrative analysis, virtual communities, and the impact of games on individuals and society. Scholars in game studies employ diverse methodologies, such as qualitative and quantitative research, ethnography, and media analysis, to explore the complexities of gaming phenomena. This research aims to deepen our understanding of the unique characteristics of games, their effects on players, and their broader cultural significance. It contributes to the development of game theory, game design principles, and informs industry practices and policies.

The team that worked on this research draws attention to how important videogames reviews are, defining them as one of the most important forms of videogaming journalism.

The goal of this research is to analyse game reviews to understand how regular players, at least those willing to write a game review, describe games, gameplay, and get their emotions influenced by the products they purchase.

By doing this, they want to gather all the information useful to evolve game design techniques.

Of course, this is a fascinating purpose, but it’s related more generally to the field of game design, which doesn’t regard just videogames, and, in addition to that, it doesn’t refer in any way to the process of videogame developing, which is the one this project wants to focus on.

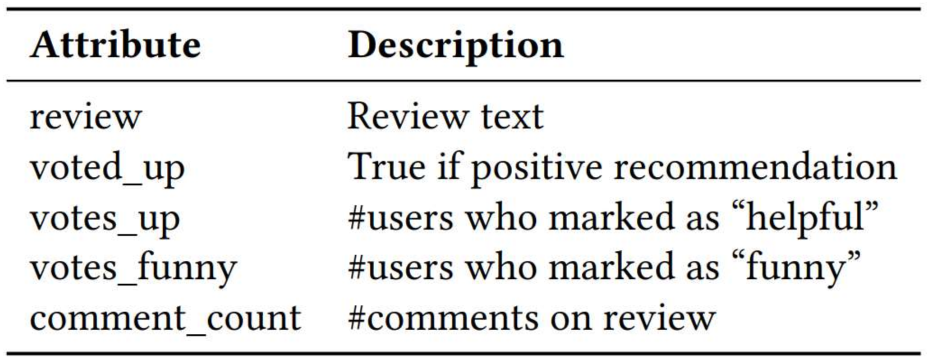
2.1.2 – “The Wisdom of the Gaming Crowd”

In the article "The Wisdom of the Gaming Crowd" [Robert Jeffrey, Pengze Bian, Fan Ji, Penny Sweetser] they will use another time videogames reviews, specifically the ones published on the well-known gaming platform “Steam” to help gamers with their future purchases and to help game developers receiving feedbacks about their products.

This team talked about how videogames, even though customers have different subjective opinions on them, must have objective properties that can be detected through the analysis of game reviews.

The problem is that, not only reviews are heavily influenced by gamers’ personal tastes, but also are often less formal, filled with humour and jokes, making even harder the analysis process.

The team, to overcome this problem, relies on the large Steam dataset, which contains well-organized reviews’ metadata, simplifying the task.



Attribute	Description
review	Review text
voted_up	True if positive recommendation
votes_up	#users who marked as “helpful”
votes_funny	#users who marked as “funny”
comment_count	#comments on review

Figure 3 - Steam API metadata [The Wisdom of the Gaming Crowd]

In order to analyse the sentiment associated with the reviews, and to recognize spam, the team used a simple Naïve Bayes algorithm to do the classification.

Not only this method is pretty much obsolete, but also this research doesn't give a concrete contribution to the field of game development.

2.1.3 – "Multilingual Persuasion Detection: Video Games as an Invaluable Data Source for NLP"

In the work "Multilingual Persuasion Detection: Video Games as an Invaluable Data Source for NLP" [Teemu Pöyhönen, Mika Hämäläinen e Khalid Alnajjar, Helsinki University, 10/07/2022], they talk about the huge potential owned by videogames dialogues, often annotated by the developers themselves, which make these textual contents definitely predisposed to be elaborated for the training of NLP models, remarking how the intersection between videogames and natural language processing isn't fully actualized yet.

The aim of this research is to build NLP models which can recognize **persuasive language**, by training them on videogame dialogues.

Persuasive language refers to the deliberate use of words, expressions, and techniques to influence and sway an audience's beliefs, attitudes, or actions. It involves employing rhetorical devices, such as emotional appeals, logical reasoning, vivid imagery, and persuasive tactics like repetition or exaggeration. Effective persuasive language aims to engage the audience, establish credibility, evoke emotions, and present compelling arguments. It is commonly used in various domains, including advertising, politics, public speaking, and marketing. By understanding and utilizing persuasive language effectively, communicators can enhance their ability to convince, motivate, and shape opinions, ultimately influencing the thoughts and behaviours of others.

To achieve this objective they used **BERT**, managing to build a multilingual persuasion detection which works in five languages: English, Spanish, French, Italian and German

The team extracted textual contents from three video games:

- Neverwinter Nights (Bioware, 2002),
- Knights of the Old Republic 1 (Bioware, 2003)
- Knights of the Old Republic 2 (Bioware, 2012).

This direction, which involves the analysis of game textual contents, may be useful for several reasons, but it is not focused on improving the quality and the variety of the contents that videogames can offer, which is what this work is aiming at.

The next paragraph will explain the second approach that is the one towards which the project is directed.

2.2 – NLP to improve videogames contents

The approach that will be used in this project is the opposite, it is about using external data sources, real or synthetic, to develop convincing NPCs that can understand human language.

2.2.1 – "Generative Agents: Interactive Simulacra of Human Behavior"

This is the direction taken by some works like "Generative Agents: Interactive Simulacra of Human Behavior" [Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, Michael S. Bernstein, 07/04/2023], in which the researchers use Large Language Models to create agents which act in a plausible way inside the game world that is set up as a real ecosystem.

In this research the team took inspiration from **sandbox games** such as "The Sims", in order to create an artificial society.

For years developers of this kind of games tried to create computational agents which can act in a plausible way in their environment and, even though they did huge progresses, these agents lack credibility.

This team, by means of the newest technologies like GPT models, managed to create a convincing ecosystem, in which every agent keeps memory of past actions and develop bonds and relations with the other agents.

The architecture developed for this work comprises three main components.

- The first is the memory stream, a long-term memory module that records, in natural language, a comprehensive list of the agent's experiences. The retrieval model combines relevance, recency, and importance to surface the records that are needed to inform the agent's moment-to-moment behaviour.
- The second is reflection, which synthesizes memories into higher level inferences over time, enabling the agent to draw conclusions about itself and others to better guide its behaviour.
- The third is planning, which translates those conclusions and the current environment into high-level action plans and then recursively into detailed behaviours for action and reaction. These reflections and plans are fed back into the memory stream to influence the agent's future behaviour.

With this architecture the researchers managed to create a system which is not only useful in the field of videogames, but also for simulation and social prototyping purposes.

2.2.2 – "Generating Role-Playing Game Quest Descriptions With the GPT-2 Language Model"

Another interesting work is "Generating Role-Playing Game Quest Descriptions With the GPT-2 Language Model" [Susanna Värtinen, Aalto University, 10/12/2021], in which they use the Language Model GPT-2, finetuned on a dataset composed of quests from various RPG videogames, to procedurally generate missions' descriptions.

This work is interesting because it applies Natural Language Processing to videogames in a different way. It's not about making characters which can act in a plausible way, it is instead related with **narrative intelligence**.

Narrative intelligence is a branch of Artificial Intelligence which refers to the ability to comprehend, generate, and engage with narratives in a meaningful way. It encompasses the cognitive processes and skills involved in understanding and creating narratives, including stories, anecdotes, and accounts of events. Narrative intelligence enables individuals to recognize patterns, interpret information, and construct coherent mental representations of narratives. It involves the integration of various elements such as plot, characters, settings, and themes to extract meaning and make sense of the narrative's structure and message.

At its core, narrative intelligence involves both the cognitive and emotional dimensions of human storytelling. It entails the capacity to perceive and analyse narrative structures, identify narrative arcs, comprehend character motivations, and recognize narrative devices like foreshadowing or symbolism. Additionally, narrative intelligence involves the ability to empathize with characters, engage with their experiences, and understand the emotional impact of the narrative.

Narrative intelligence extends beyond individual interpretation; it also includes the capacity to construct narratives and effectively communicate them to others. This encompasses the skills to create engaging storylines, develop well-rounded characters, and evoke emotional responses from the audience. Narrative intelligence is crucial in a range of domains, including literature, film, theater, marketing, and even everyday communication. It plays a significant role in shaping our understanding of the world, conveying cultural values, and fostering social connections.

Studying narrative intelligence offers insights into how humans process and derive meaning from narratives, both in personal and social contexts. By exploring the intricacies of narrative construction and interpretation, researchers can deepen our understanding of human cognition, emotions, and the powerful role that narratives play in shaping our perceptions and behaviors.

This project, in fact, aims to train generative models that can create imaginative RPG quests.

The model used by the authors is Generative Pre-trained Transformer 2 (GPT-2), which is finetuned on a publicly available data set of 978 quests from six different RPG games.

Some of the quests generated after the fine-tuning might not be able to convince a human reader, but still this work about computational creativity managed to achieve pretty good results.

These research, although valuable, don't give support to videogame developers in the process of applying NLP to videogames, they show instead how to create a new kind of smart videogames with a certain knowledge in the field of Artificial Intelligence, which is not available for all the programmers out there who want to make their games.

This project instead aims to define a workflow, which should be easy and fast, to create new types of videogames that get rid of all those kinds of pre-programmed dialogues to which we are so accustomed by now.

The ways of dealing with this purpose will be covered in the next chapters.

2.3 – Commercial side

Now let's put aside the research done. Game studios recently started to develop commercial games which use the NLP technology.

Let's look at one example.

2.3.1 – THE PORTOPIA SERIAL MURDER CASE – Square Enix AI

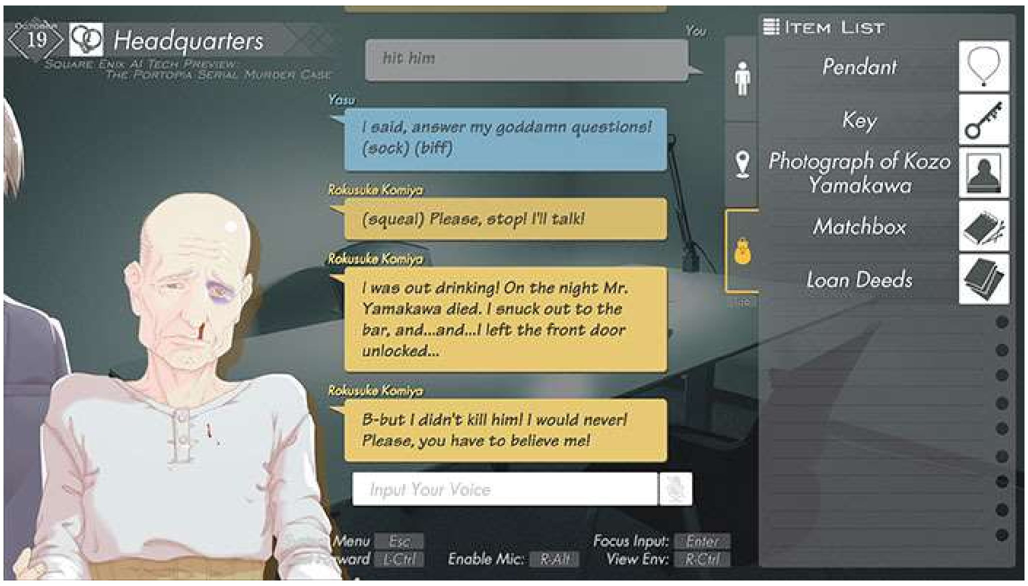
The AI division of Square Enix recently published an educational tech demonstration of Natural Language Processing applied to a videogame.

The game is called "The Portopia Serial Murder Case" and it's an evolved version of the homonymous game developed by Yuji Horii and first published in Japan in 1983, which was similar to a **text adventure**.

Text adventures emerged in the late seventies as one of the earliest forms of computer games. They rely primarily on textual descriptions to present a virtual world to players, who interact with the game by typing commands and receiving text-based responses. Inspired by tabletop role-playing games, text adventures offer immersive storytelling and puzzle-solving experiences. Early examples like "Colossal Cave Adventure" and "Zork" gained popularity on mainframe and home computers. While graphics-based games eventually became dominant, text adventures continue to have a niche following, with modern adaptations and a dedicated community of enthusiasts. They showcase the enduring appeal of engaging narratives and imaginative gameplay driven by text-based interaction.

This remake wants to overcome several limitations related to old text adventures, which didn't have access to modern technologies and, due to this, could often be frustrating for the players, because even if users knew which action they wanted to perform, they were unable to do so because they could not find the right wording.

The developers decided to put aside the game functionalities that make use of text generation, because they risk leading to unethical replies.



You: hit him

Yasu: I said, answer my goddamn questions! (sock) (biff)

Rokusuke Komiya: (squeal) Please, stop! I'll talk!

Rokusuke Komiya: I was out drinking! On the night Mr. Yamakawa died. I snuck out to the bar, and...and...I left the front door unlocked...

Rokusuke Komiya: B-but I didn't kill him! I would never! Please, you have to believe me!

Figure 4 - Game screenshot [https://www.jp.square-enix.com/ai-tech-preview/portopia/en/]

CHAPTER 3 – Methodology

3.1- Artificial intelligence

Before starting, is important to talk about what Artificial Intelligence is.

Artificial Intelligence (AI) is a field of computer science that focuses on creating intelligent machines that can mimic human-like cognitive abilities. It involves developing computer programs and algorithms capable of processing information, learning from it, and making decisions or predictions. These intelligent systems can perform tasks such as speech recognition, image classification, problem-solving, and even autonomous decision-making. AI has wide-ranging applications in various industries, including healthcare, finance, transportation, and entertainment, revolutionizing the way we live and work.

Now, Artificial Intelligence has several sub-fields, which in most of the cases can be recursively split into other chunks.

3.2 – Machine Learning

This project will take into account one specific sub-field of AI, which is **machine learning**.

Machine learning is a branch of artificial intelligence that involves developing algorithms and models that allow computers to learn from data and make predictions or decisions without explicit programming. It is based on the idea that machines can automatically improve and adapt their performance by analysing patterns and extracting meaningful insights from large amounts of data.

In machine learning, models are trained using data to recognize patterns and make predictions or classifications on unseen data. This process involves identifying features or attributes in the data that are relevant to the task at hand. The models can then use these learned patterns to make accurate predictions or decisions on new data.

To train a model, which is the practice of learning through data, you need in fact to identify **input features**, which are the features that the model will receive when a prediction is needed, and the **output features**, which are what you expect to receive when asking something to the model.

The combination of input features and output features is often referred to as **"example"**.

Machine learning techniques include **supervised learning**, where models learn from labelled data, and **unsupervised learning**, where models discover patterns and structures in unlabelled data. Additionally, there are reinforcement learning approaches, where models learn through interaction with an environment, receiving rewards or penalties based on their actions.

In this research we will only deal with supervised learning.

The quality and the performances of a machine learning model, as may appear clear from the previous description, depend on several factors, which will be soon analysed.

3.2.1 – Learning data

Since machine learning models learn through a large amount of data, the quality of the dataset provided is essential to reach good performances.

The data provided should be various and the number of examples can't be defined in advance, it depends on the specific model used.

A complex model generally requires more data to be trained than a simple one, so we can say that the number of examples given should be weighted based on the complexity of the model.

Unlike what could sound reasonable, there is a problem even with choosing an amount of data which is too large. Although a small number of examples could lead to a model with poor performance, if the amount of data is too high compared to the complexity of the model it could lead to the problem of **overfitting**.

Before we can talk about this problem, we should introduce details on how datasets are organized.

Examples in datasets are typically divided into three subsets: **train set**, **validation set**, and **test set**. These subsets have different purposes within the process of learning.

1. **Train Set**: The training set is the largest subset of the dataset and is used to train the model. It consists of input examples paired with their corresponding target variables (in supervised learning). The model learns from this data by adjusting its internal parameters to minimize the error between predicted and actual values. The training set helps the model capture the underlying patterns and relationships in the data.
2. **Validation Set**: The validation set is used to fine-tune the model and select the best **hyperparameters** or model architecture. It acts as a proxy for unseen data and helps assess the model's performance and generalization ability. The model's performance on the validation set provides insights into its ability to handle new data and helps in making decisions about hyperparameter tuning or model selection. The validation set helps prevent overfitting, where the model becomes too specialized to the training data and performs poorly on new data.
3. **Test Set**: The test set is used to evaluate the final performance of the trained model. It represents unseen data that the model has not been exposed to during training or validation. The test set provides an unbiased estimate of the model's performance on new, real-world data. Evaluating the model on the test set helps determine its effectiveness and generalization ability.

It is important to maintain the independence of these subsets. Therefore, the test set should not be used for any model selection or hyperparameter tuning decisions. Instead, the test set should only be used for the final evaluation of the model's performance.

By using separate training, validation, and test sets, machine learning practitioners can develop models that generalize well to new data and have a more reliable estimate of their performance on unseen examples.

We will talk in a while about what hyperparameters are.

Now we explained all the concepts needed to talk about the overfitting.

Overfitting is a common problem in machine learning where a model becomes overly complex and learns to fit the training data too closely, resulting in poor generalization to unseen data. It occurs when the model captures noise or irrelevant patterns in the training set that do not exist in the underlying data distribution. As a result, the model performs well on the training data but fails to generalize to new examples. Overfitting can be mitigated through techniques such as regularization, cross-validation, early stopping, or increasing the size of the training dataset.

Incredibly high accuracy metrics on the train set could be indications of overfitting.

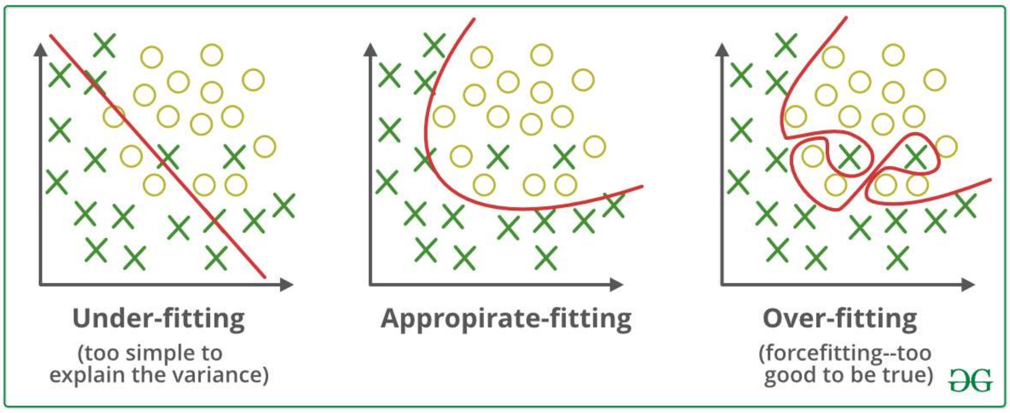

Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)

Figure 5 - [https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/]

3.2.2 – Pre-processing

An important aspect to consider when dealing with Machine Learning is related to the format that data should have to be processed by the training algorithm.

The sequence of operations that are applied to data before the execution of the training process is called **pre-processing**.

More in details, pre-processing in machine learning refers to the preparation and transformation of raw data before it is used to train or apply a machine learning model. It involves a series of steps such as data cleaning, feature selection or extraction, scaling, and normalization.

This mixture of operations aims to enhance the quality and relevance of the data, ensuring it is in a suitable format for the chosen machine learning algorithm. By pre-processing the data, irrelevant or noisy information can be removed, features can be standardized, and missing values can be handled. Effective pre-processing plays a crucial role in improving the accuracy, efficiency, and reliability of machine learning models.

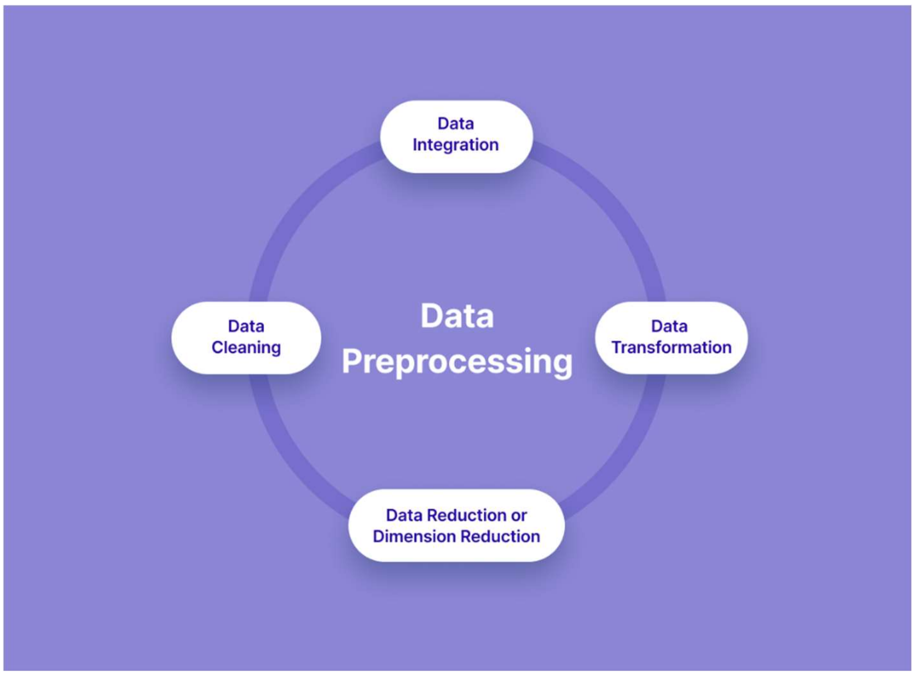


Figure 6 – [https://www.v7labs.com/blog/data-preprocessing-guide]

3.2.3 – Hyperparameters

In machine learning, **hyperparameters** are configuration settings that are external to the model and are set before the learning process begins. Unlike model parameters, which are learned from the data during training, hyperparameters are not updated based on the training data but rather act as controls that influence the learning process and model behaviour.

Hyperparameters define the characteristics of a model and can include parameters such as learning rate, regularization strength, number of hidden layers in a neural network, or the depth of a decision tree. They determine how the model learns and generalizes from the data.

Selecting appropriate hyperparameter values is crucial as it affects the model's performance, convergence speed, and generalization ability. Hyperparameters are usually tuned through techniques like grid search or randomized search, where different combinations of values are evaluated based on performance metrics using cross-validation.

It is important to note that hyperparameter tuning is an iterative process, and selecting the optimal hyperparameters often involves trade-offs. The choice of hyperparameters depends on the specific dataset, problem complexity, computational resources, and desired trade-off between model complexity and generalization.

Hyperparameters have a significant impact on the behaviour and performance of machine learning models, and finding the right combination can significantly enhance model performance and generalization across different datasets and problem domains.

After explaining these concepts it might be clear that to develop a system that uses machine learning a lot of choices must be done, but now let’s deal with an important problem.

The goal that this project wants to reach is to create NPCs in a videogame that can understand the sentences of the player, and we clarified that the means that we will use is machine learning. Since a large amount of data is required to train this kind of models, how can we manage to gather examples on such a specific task which is related to the behaviour of a specific character?

The solution that will be adopted in this work is related to **synthetic data**.

3.2.4 – Synthetic data

Synthetic data in machine learning refers to artificially generated data that mimics the patterns and characteristics of real-world data. It is created using algorithms or statistical models rather than being collected from actual observations. Synthetic data is used when obtaining real data is challenging or limited, or when privacy concerns restrict the use of sensitive data. It can be helpful in augmenting training datasets, exploring different scenarios, testing model performance, or addressing data imbalance. Generating synthetic data requires careful consideration to ensure that it adequately represents the underlying distribution and preserves key statistical properties of the real data.

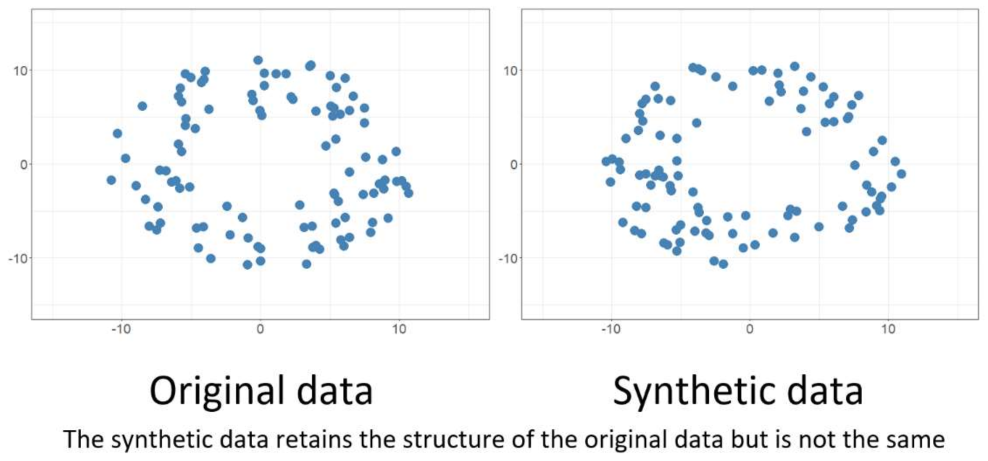


Figure 7 - [https://dataingovernment.blog.gov.uk/2020/08/20/synthetic-data-unlocking-the-power-of-data-and-skills-for-machine-learning/]

In this specific case is not possible to gather data from real word's services, since they obviously can't cover the multitude of imaginary situations with which a videogame can deal with.

3.3 – Deep learning

As we have previously stated, machine learning is a vast subject, so we need to be more detailed.

Specifically, this work will deal with a sub-field of machine learning which is **Deep learning**.

Deep learning focuses on training specific machine learning models which are referred to as **artificial neural networks** to learn and make predictions. It is inspired by the structure and function of the human brain. Deep learning involves creating complex networks of interconnected artificial neurons, known as deep neural networks, which can learn and extract hierarchical representations from data.

Unlike traditional machine learning, deep learning models can automatically learn hierarchical representations of data at multiple levels of abstraction. This allows them to recognize and understand intricate patterns and relationships in the data. Deep learning excels in tasks such as image and speech recognition, natural language processing, and even playing complex games.

Deep neural networks consist of layers of interconnected neurons, with each neuron performing simple computations. The networks learn by adjusting the connections between neurons based on training data, optimizing their performance through a process called backpropagation.

Deep learning has shown remarkable success in various domains, revolutionizing fields like computer vision, voice assistants, autonomous driving, and healthcare diagnostics. Its ability to learn and extract complex features from large amounts of data has made it a powerful tool in artificial intelligence.

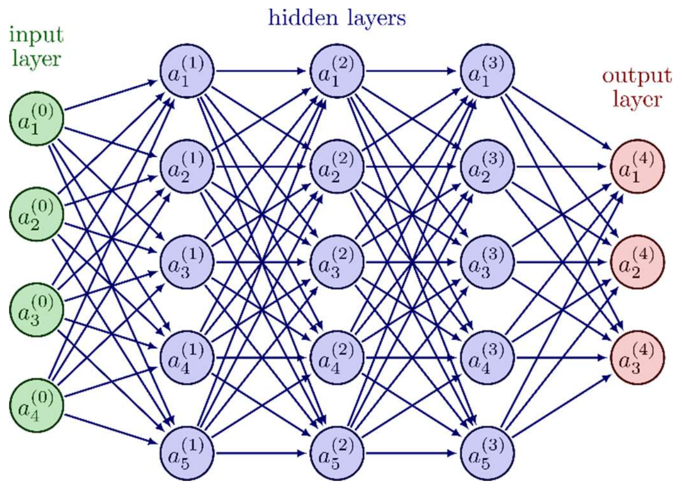


Figure 8 - Visual representation of a Neural Network [https://tikz.net/neural_networks/]

But how does the training process of neural networks work?

A neural network can be represented as a weights' matrix, in which every weight, given the input, contributes to define the final output.

The training of neural networks involves iteratively adjusting the weights and biases of the network to minimize the errors between the predicted output and the actual target output. These errors contribute to define a **loss function** which is a cost metric that depends on weights and biases of the neural network. The more are the errors, the more the loss function grows.

Said that, the essential problem of training a neural network is reduced to a search for the minimum of a multivariable function.

So, how does the training process concretely work?

To explain this, we must resort to the mathematical concepts of **derivative** and **gradient**.

In analysis, a **derivative** represents the rate of change of a function with respect to its independent variable. It measures how a function's output changes as its input varies. The derivative of a function at a specific point gives the slope of the tangent line to the graph of the function at that point. It provides valuable information about the function's behaviour, such as whether it is increasing or decreasing, concave up or down. The derivative is defined as the limit of the difference quotient as the interval between two points approaches zero.

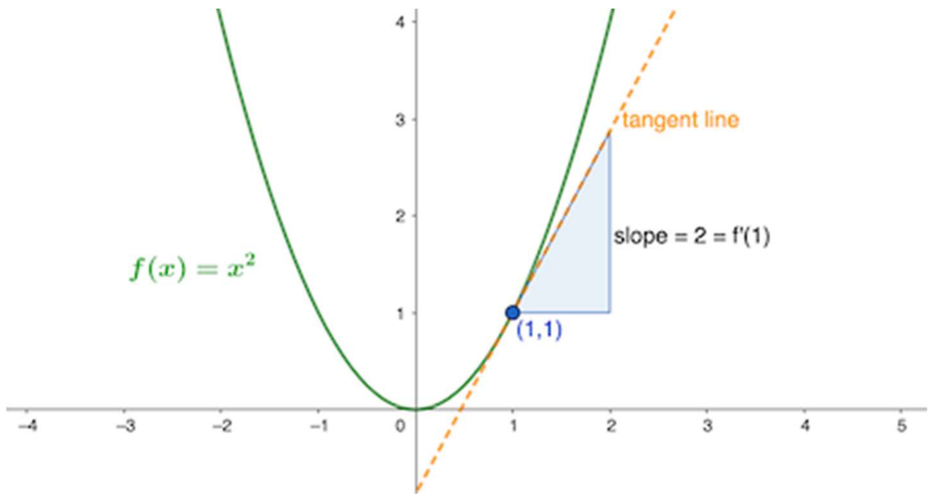


Figure 9 - Example of derivative [https://study.com/learn/lesson/data-mining-identifying-functions-from-derivative-graphs.html]

The **gradient** instead represents the vector of partial derivatives of a multivariable function. Like in the one-dimensional case, with a single derivative, the gradient can be used to find the minimum of a multivariable function, leading to the **gradient descent algorithm**, which is comparable with following the slope of the graph to reach the min.

Now that we have introduced these fundamental concepts, the explanation about how the training of neural networks works will be clearer.

The process begins by initializing the network's parameters randomly. The training data is then fed into the network, and the predicted outputs are compared with the true outputs. Through a process called backpropagation, the network computes the gradients of the error with respect to the parameters, which indicate how the parameters should be updated. These gradients are then used to update the weights and biases using an optimization algorithm, such as gradient descent. This iterative process continues until the network converges to a state where the error is minimized.

3.3.1 – Deepening of gradient descent algorithm

More formally, the **gradient descent algorithm** is an iterative optimization technique used to minimize a function.

In a simple way this algorithm works like this:

1. Begin with an initial guess for the function's parameters.
2. Calculate the gradient of the function at that point. The gradient indicates the direction of steepest descent.
3. Take a step in the opposite direction of the gradient (the direction of descent) and update the parameters accordingly. The size of the step is controlled by a learning rate.
4. Repeat steps 2 and 3 until convergence or a stopping condition is met. Convergence occurs when the changes in parameters become small or the error is minimized.
5. The algorithm aims to iteratively adjust the parameters, moving towards the function's minimum. By following the negative gradient, it descends closer to the optimal solution.

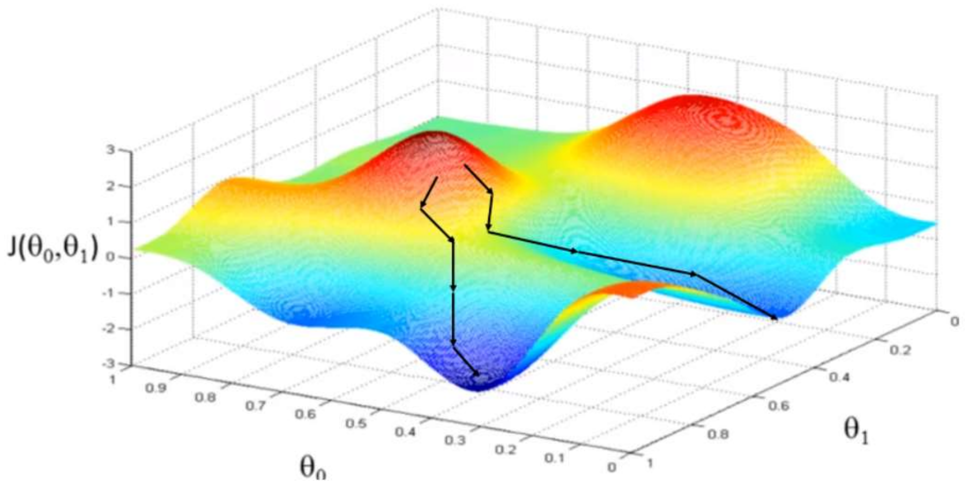


Figure 10 - Visual representation of the Gradient Descent Algorithm [https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/]

3.4 – Choices for the project

Given this overview on all the main topics that this project wants to face, it will be clear that there are a lot of considerations to take into account to develop a software that aims to link two huge fields like Artificial Intelligence and videogames.

First of all, I must focus on the needs of the game developers and the requirements which are necessary to create NLP models which don't come into conflict with the game rules.

3.4.1 – Text generation models

I must, by now, exclude a certain type of models, which are **text generation models**.

A text generation model refers to a type of NLP model that is designed to generate coherent and contextually relevant text based on a given input or prompt.

Text generation models typically utilize large amounts of text data, to learn the statistical relationships and patterns present in the language. They employ algorithms like recurrent neural networks (RNNs, a specific type of Neural Network) or transformer models, such as the popular GPT (Generative Pre-trained Transformer) architecture, to capture the dependencies between words or phrases in a given text.

During the training process, the model learns to predict the probability distribution of the next word or sequence of words based on the context provided by the input data. It leverages this learned knowledge to generate new text by sampling words or phrases that are most likely to follow the given context.

Text generation models have demonstrated impressive capabilities in various applications. They can be used to automatically generate human-like responses in chatbots, assist in language translation, summarize documents, create personalized recommendations, generate creative written content, and even aid in storytelling or dialogue generation for video games and virtual environments, so they are even related to the field of computational creativity.

It's important to note that text generation models are not flawless and may sometimes produce text that is nonsensical, biased, or inappropriate. Ethical considerations and careful evaluation of the generated output are crucial when using these models in real-world applications. Ongoing research and development efforts in the field of text generation aim to improve the quality, diversity, and controllability of generated text.

But there is another important reason that made me decide to give up on these models (other than lack of time), which is related to the specific application field, the one of videogames.

Text generation models, in fact, due to their unpredictability, could be really hard to handle when dealing with a game world which has specific balance.

A model capable of generating text with an incredibly high level of randomness, if not managed properly, could inevitably break game rules.

Developing a framework that works with this technology, due to these reasons, could take a considerable amount of time, so even though text generation models could give a higher degree of freedom and immersivity to the player, I decided to set them aside for now.

That being said, without text generation, the point left to focus on is the understanding of the sentences that a player can say.

The natural language is as fascinating as complex, there are a lot of ways to express the same concept, even in the same language. Every human decides to use a specific expression based on several factors which can be influenced by culture, context, emotions, and others…

On the other hand, computers need rigid instructions in order to execute specific functionalities, that’s why it’s hard to put in communication humans and machines.

Nowadays practically all games, due to the problems that we stated before, give to the player a set of pre-determined sentences to communicate their intentions to the NPCs. This is what I want to work on.

To solve this problem, we must resort to specific Machine Learning models that can learn through data how to associate values to sentences.

3.4.2 – Classifiers and regressors

There are two types of models that can be used for this purpose: **classifiers** and **regressors**.

In machine learning, a **regressor** is a type of predictive model used for solving regression problems. Regression involves predicting a continuous value or quantity based on input features. A regressor learns the underlying patterns and relationships within a dataset to make predictions on new, unseen data.

Regressors employ various algorithms such as linear regression, decision trees, support vector regression, or neural networks. These algorithms analyse the relationships between input features and the target variable to create a function that maps the input features to the predicted continuous output.

Once trained, the regressor can take input features and produce predictions for the target variable. The quality of the predictions is evaluated using metrics such as mean squared error, mean absolute error, or R-squared.

It's important to select an appropriate regressor algorithm and tune its parameters to achieve the best performance for a specific regression problem. The choice of regressor depends on the characteristics of the dataset, the complexity of the relationship between features and the target variable, and the desired interpretability or flexibility of the model.

A **classifier** instead is a type of predictive model used for solving classification problems. Classification involves assigning predefined categories or labels to input data based on their features. Classifiers learn from labelled training data to make predictions on new, unlabelled data.

Classifiers employ various algorithms such as logistic regression, decision trees, support vector machines, or deep neural networks. These algorithms analyse the patterns and relationships between input features and the corresponding class labels to create a decision boundary or a probability distribution.

Classifiers and regressors can generally be implemented to work with every sort of input features, like images, sounds, and others, but in this case, of course, the input features desired will be the words of a sentence.

Even though regressors could implement cool functionalities, like text scoring which can be really useful in the field of games, I decided to mainly use classifiers for this project, because they are easier to control in this specific context.

In future I will still be able to integrate regressors.

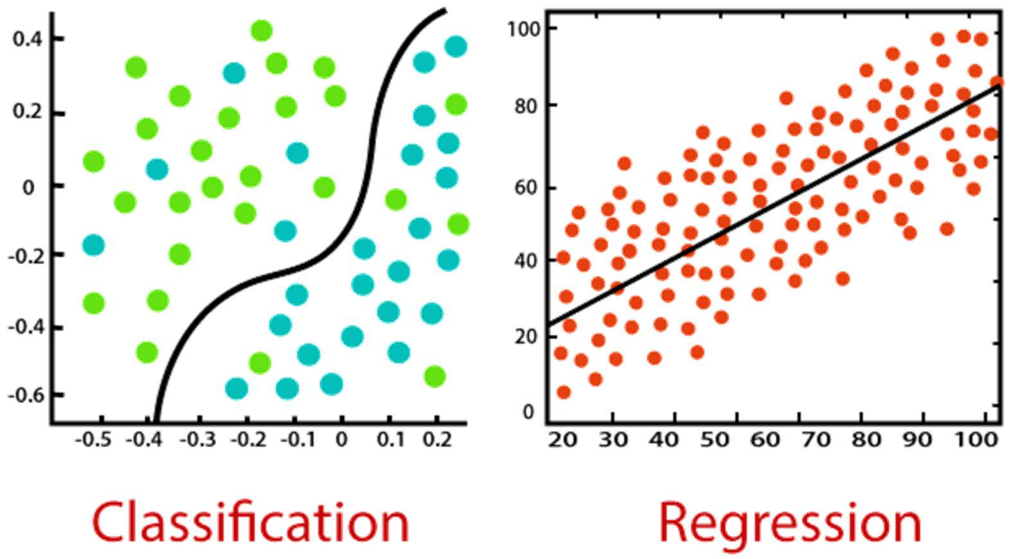


Figure 11 - Classification vs Regression [https://www.javatpoint.com/regression-vs-classification-in-machine-learning]

3.4.3 – Types of Classification

Classification can be split into two different types, which are **binary classification** and **multi-class classification**.

- **Binary Classification**: this type of classification is applied to those cases in which there are only two labels or classes.
  Typically, this kind of classification involves a positive class, which represents somehow the “normal state” and the negative class, which, on the contrary is associated to the “abnormal state”.

  The way to solve this type of classification is to find the **decision boundary**, which is the line, curve, or hyperplane that best separates the examples on a graph.

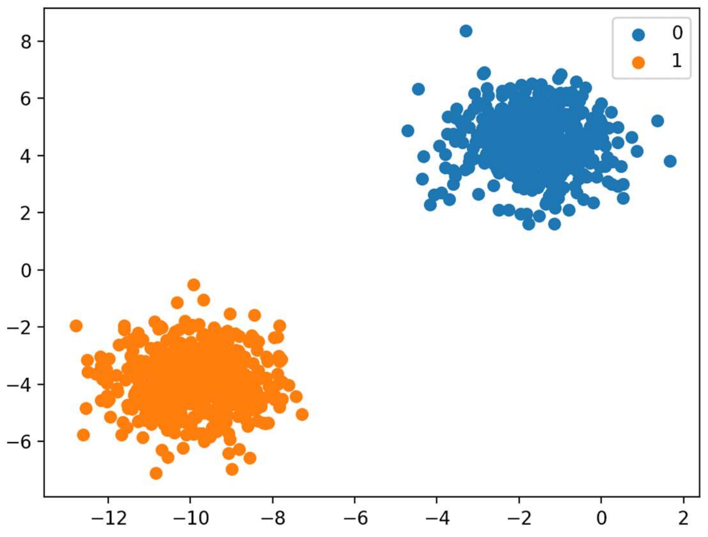


Figure 12 - Visual representation of binary classification [https://machinelearningmastery.com/types-of-classification-in-machine-learning/]

Binary classification can be used for several tasks, here are some examples:

- Spam Filtering: This type of classification can be used to classify incoming emails as either spam or non-spam based on their content, subject, sender, and other features. This helps in filtering out unwanted emails and keeping the inbox clean.
- Fraud Detection: Binary classification is utilized to identify fraudulent activities in financial transactions. By analysing patterns and features in transaction data, a classifier can flag suspicious transactions, helping to prevent fraud and secure financial systems.
- Credit Scoring: Binary classification is used in the finance industry to predict the creditworthiness of individuals. By analysing various factors such as income, credit history, employment status, and demographic information, a classifier can determine whether a person is likely to default on a loan or has a low risk of default.
- Sentiment Analysis: Binary classification is employed in natural language processing to determine the sentiment of textual data, such as social media posts, product reviews, or customer feedback. It can classify text as positive or negative, indicating the sentiment expressed by the author.
- Disease Diagnosis: In healthcare, binary classification can be applied to diagnose diseases based on patient symptoms, medical tests, or imaging data. For example, a classifier can determine whether a patient has a certain medical condition or not, aiding in timely treatment and intervention.
- Intrusion Detection: Binary classification is employed in cybersecurity to detect network intrusions or malicious activities. By analysing network traffic patterns, system logs, and behaviour, a classifier can identify whether an activity is normal or indicative of a security breach.

These are few examples, of course the number of actual applications is way higher.

- Multi-class classification: this type of classification, as may be clear from the name, refers to the classification tasks which have more than two labels; unlike binary classification, in multi-class classification there are no notions about positive or negative classes.

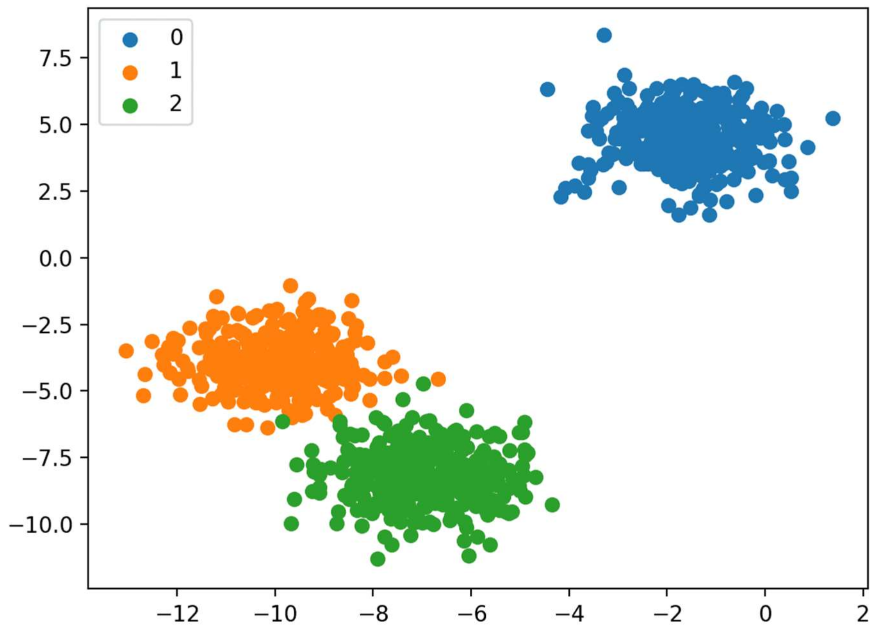


Figure 13 - Visual representation of multi-class classification [https://machinelearningmastery.com/types-of-classification-in-machine-learning/]

Even in this case the number of applications is huge, but we will mention only few examples:

- ○ Text classification: is a task which involves assigning predefined classes to text documents based on their content.
- ○ Intent classification: is a sub-task of text classification, and its aim is to find the intention behind a sentence.
- ○ Sentiment analysis: in this case, it involves more than two sentiments (positive or negative).

3.4.4 – Classification tasks needed for this project

After describing a minimum part of the tasks in which classification is involved, we have to focus on the ones of our interest.

- **Intent classification**: the intent classification is essential to understand the communicative intentions of the players without resorting to the use of pre-programmed dialogs or key words, which could lead to poor flexibility and variety of gaming styles, leading to a gameplay which fits perfectly to every player.
- **Sentiment analysis**: videogames are strongly related to the emotions of the players. when gamers are playing, they could pass through a wide range of emotions, frustration when they don't manage to achieve the expected goals, excitement when they access to a new part of the game with their improved skills, or boredom when the challenge level is too low. Is clearly important to understand the sentiment of the player in order to eventually adapt the gameplay, the difficulty and other aspects according to the feelings detected. That's why sentiment analysis is a component needed to improve game contents.
- **Named Entity Recognition**: Named Entity Recognition (NER) is a natural language processing task that involves with identifying and classifying named entities such as person names, locations, organizations, and dates in text. It helps in extracting valuable information and understanding the structure and meaning of textual data. This kind of task is essential to recognize the entities involved in a sentence, in order to perform actions such as picking items, giving instructions about locations or directions and many others.

3.4.5 – Possible classifiers implementations

There are many possible implementations to build a text classifier. Here are some examples.

- **Rocchio classifier**: The Rocchio classifier is a supervised machine learning algorithm used for text classification tasks. This implementation classifies new instances by comparing their feature vectors to representative prototypes of each class. The Rocchio classifier is a simple and interpretable algorithm, suitable for text classification tasks where feature vectors representing text instances can be computed and compared using a similarity metric. Even considering these positive aspects, this implementation is obsolete, and doesn't allow to create smart models since it doesn't learn context, due to the text representation used, the **bag of word (BOW)**, which doesn't preserve the order of the words in a text.
- **The Naive Bayes classifier**: this is a probabilistic machine learning algorithm widely used for text classification and spam filtering. It's based on Bayes' theorem, assuming that the features are conditionally independent given the class label. Even though this implementation is fast, has a good scalability and due to its probabilistic nature gives interpretable results has some big problems:
    - Independence Assumption: The "naive" assumption of feature independence may not hold in all cases, potentially affecting classification accuracy when there are strong correlations among features.
    - Limited Expressiveness: Naive Bayes may struggle with complex relationships and subtle interactions among features that require more sophisticated models.
- **Neural Networks**: Neural networks can be used to perform text classification. Even though they're harder to train, they guarantee incredible performances, since they can learn the contextual relations between words in a sentence.

Classification with neural networks is the direction chosen for this work.

But now, even after I decided to build classifiers with neural networks, another important choice must be made. Neural networks, in fact, could be implemented with several different architectures, which have been developed and have evolved through the years.

In order to choose a specific architecture is important to know some details about them.

3.4.6 – Neural Network's architecture

1. **Feedforward Neural Networks (FNNs)**: Also known as Multi-Layer Perceptrons (MLPs), FNNs consist of an input layer, one or more hidden layers, and an output layer. Information flows in a forward direction, and each neuron in a layer is connected to all neurons in the subsequent layer.
2. **Convolutional Neural Networks (CNNs)**: CNNs are commonly used for image and video processing tasks. They have convolutional layers that apply filters to input data, capturing spatial relationships and extracting meaningful features. Pooling layers reduce dimensionality, and fully connected layers make final predictions.
3. **Recurrent Neural Networks (RNNs)**: RNNs are designed to handle sequential data, such as natural language processing and time series analysis. They have feedback connections that allow information to persist and flow from previous steps, enabling memory of past information.
4. **Long Short-Term Memory Networks (LSTMs)**: LSTMs are a type of RNN architecture that address the vanishing gradient problem and are particularly effective in capturing long-term dependencies. They utilize memory cells to selectively remember or forget information over time.
5. **Gated Recurrent Units (GRUs)**: GRUs are another variant of RNNs that also tackle the vanishing gradient problem. They have gating mechanisms that control the flow of information, similar to LSTMs but with a simplified architecture.
6. **Autoencoders**: Autoencoders are unsupervised learning models that aim to learn compressed representations of input data. They consist of an encoder network that maps input data to a latent space and a

decoder network that reconstructs the input data from the latent space.

7. **Generative Adversarial Networks (GANs)**: GANs are comprised of two networks: a generator network that generates new data samples and a discriminator network that tries to distinguish between real and generated samples. GANs are commonly used for generating realistic synthetic data.

These are just some examples of neural network architectures, of course there are many more specialized architectures and variations developed for specific tasks or to address specific challenges.

Initially I decided to work with Long Short-Term Memory Networks, since they work well with sequential data, like words in a sentence, without running into the problem of the vanishing gradient or, at least, reducing it.

The **vanishing gradient problem** refers to the issue in deep learning where the gradients used to update neural network weights become extremely small during backpropagation, leading to slow or no learning in early layers.

My relator, Professor Pierpaolo Basile, suggested me a more modern architecture, which guarantees better performance than the previously mentioned models, the **Transformers architecture**.

The **transformer architecture** is a deep learning model that revolutionized natural language processing (NLP) tasks. It was introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017. Unlike traditional sequential models such as recurrent neural networks (RNNs), transformers utilize a self-attention mechanism to capture dependencies between words in an input sequence.

The transformer architecture consists of an encoder and a decoder. Each layer in the encoder and decoder contains multi-head self-attention and feed-forward neural network components. Self-attention allows the model to weigh the importance of different words when generating representations, capturing global dependencies efficiently. Positional encoding is added to the input sequence to convey the order of the words.

Transformers have been successful in various NLP tasks such as machine translation, question-answering, and language generation. They have several advantages over RNNs, including parallelizability, the ability to capture long-range dependencies, and reduced vanishing or exploding gradient problems. Transformers have become a cornerstone in modern NLP and have influenced advancements in other domains of deep learning beyond language processing.

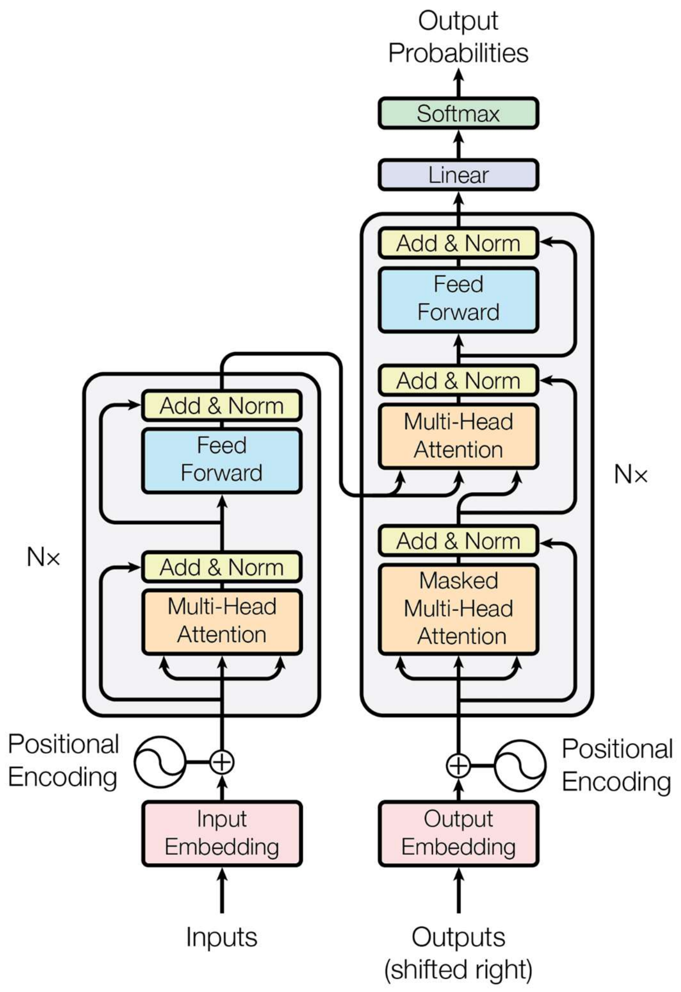


Figure 14 - Transformers architecture [Attention Is All You Need]

3.4.7 – Entities organization

The tool I want to develop not only must help to recognize intent, sentiment and entities related to a sentence, but also must organize those entities in a smart way.

Videogames often have a worldbuilding which involves several imaginary or realistic entities which have to be manipulated to achieve some goals.

In order to build models that can recognize and act on those entities properly, a good organization is needed.

I took inspiration from **WordNet** for this purpose.

WordNet is a lexical database and ontology widely used in natural language processing and computational linguistics. It organizes words into synsets (sets of synonyms) and defines semantic relationships between them. The organization of WordNet follows a hierarchical structure known as an ontology.

At the top level, WordNet is divided into four major semantic categories: nouns, verbs, adjectives, and adverbs. Each category is further organized into synsets representing specific concepts or meanings. Synsets are interconnected through various relationships, including hyponymy (is-a), hypernymy (has-a), meronymy (part-of), and antonymy (opposite).

Within each synset, words with similar meanings are grouped together. For example, in the noun category, the synset "car" may include words such as "automobile," "vehicle," and "motorcar." Each synset is assigned a unique identifier and includes a definition to clarify its meaning.

WordNet provides a rich network of relationships that capture semantic associations between words. These relationships enable tasks such as word sense disambiguation, semantic similarity calculations, and semantic search.

Additionally, WordNet contains additional linguistic information, such as verb frames and adjective positions, which help capture the syntactic behaviour and collocational patterns of words.

Overall, WordNet's ontology organizes words into hierarchies and defines relationships between them, facilitating the exploration and understanding of semantic connections in language.

Inspired by this ontology, I decided to organize game entities in a hierarchical way, using a tree in which every node corresponds to a multilingual synset, in order to create a simple but smart organization system, in which is possible to apply different metrics to perform tasks like world sense disambiguation (e.g., Leacock-Chodorow similarity, …).

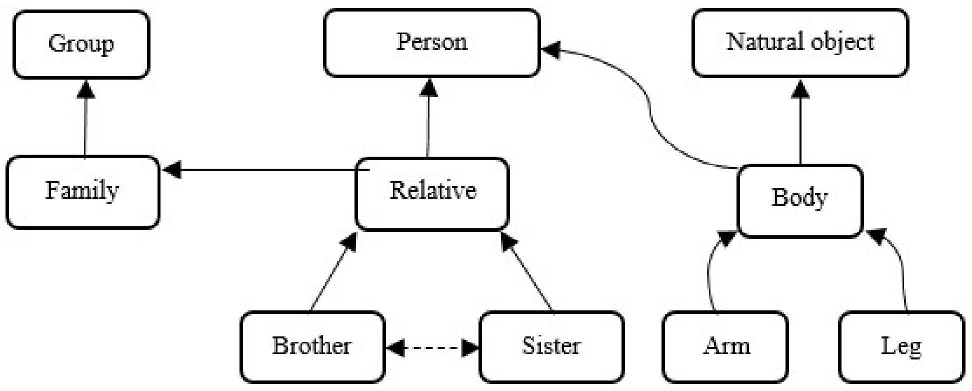


Figure 15 - Example of wordnet structure [https://www.researchgate.net/publication/340694384_Word_Sense_Disambiguation_by_Context_Detection/figures?lo=1&utm_source=google&utm_medium=organic]

CHAPTER 4 – Implementation

Now that we have deepened the knowledge about the theorical aspects and the methodology, we can talk about the implementation details of this project.

As previously stated, with this work I want to define a tool which can be used in a simple way by game developers, and which should interact with the videogames in an efficient way.

We identified the tasks that we need to implement in order to make players sentences understandable to the system, which are:

- Intent Classification
- Sentiment analysis
- Named Entity Recognition

To perform these tasks, I implemented two general types of classifiers:

- Classifiers at the level of sentence: classifiers that assign a class to the whole sentence. These models can be used for sub-tasks like Intent Classification and Sentiment Analysis.
- Classifiers at the level of words/tokens: classifiers that associate a class to every single word/token in the sentence. This kind of models can be used for Named Entity Recognition (NER) and Part of Speech (POS) Tagging.

These two classifiers, combined, can easily be used, and eventually chained to analyse player’s sentences in several ways.

A simple possible pipeline could be the following:

- Player’s sentence is given to an Intent Classification model.
- Based on the intent recognized, the system could both
    1. Directly execute the corresponding action if no other parameters are required.
    2. Pass the sentence to a NER model which can recognize the entities involved in the action, including their corresponding role.

Clearly, according to the needs, the pipeline could get more complex, in order to analyse other parameters like user’s sentiment, which can be used to implement some innovative functionalities.

I deepened my knowledge on the topic, in order to understand which are the best practices to implement these classifiers and in the end I decided to use **Bidirectional Encoder Representations from Transformers (BERT)**.

4.1 – BERT

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing model. It revolutionized the field by introducing bidirectionality and pretraining on large corpora, enabling it to capture contextual word embeddings. BERT employs a transformer architecture, which allows efficient parallelization and modelling long-range dependencies. It excels in various language understanding tasks, including sentiment analysis, named entity recognition, and question answering. Fine-tuning BERT on specific downstream tasks has yielded remarkable performance improvements. BERT's impact has been profound, influencing subsequent research and leading to advancements in numerous NLP applications.

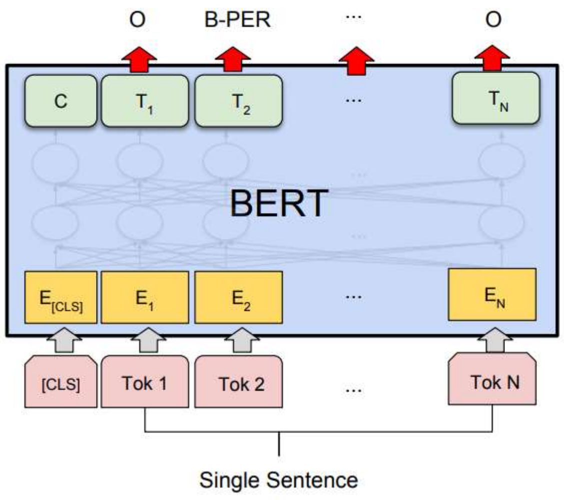


Figure 16 - BERT architecture [https://yashuseth.wordpress.com/2019/06/12/bert-explained-faqs-understand-bert-working/]

Architectures based on Transformers, like we previously stated, overcome several limitations imposed by LSTMs.

LSTMs have these following problems:

- Slow to train: the inputs are processed sequentially, so the training consumes a lot of time.
- Can't understand the contextual relation between words in a sentence: even if you consider bidirectional LSTMs, they just analyse separately the sentence in two directions.

Transformers instead are:

- Fast: they process words simultaneously, reducing the time needed to train them.
- The context is preserved: Transformers can learn context from both directions simultaneously.

These are the reasons why I decided to use this architecture to develop these classifiers.

4.2 – Back and Front splitting

Another important problem to face is about how to make this architecture compatible with all game engines, which is the most important part since I want this tool to be completely independent of the specific language or framework. In addition to that, this software should allow the game developers who have a poor knowledge about Artificial Intelligence, to train their models in the easiest way.

I decided to not implement this tool as a library or framework, because that would limit its portability, instead I decided to develop it as an independent software.

In order to do this I split the tool, which I decided to call **AIlve**, into two different parts, **back end** and **front end**.

The practice of splitting a software into back end and front end is a common approach in software development. It involves dividing the responsibilities of a software application into two main parts.

The back end refers to the behind-the-scenes operations, such as data processing, business logic, and database management. It handles the core functionality of the application and interacts with the front end.

On the other hand, the front end focuses on the user interface and user experience. It deals with the presentation layer, including visual design, user input, and displaying information to users.

By separating the back end and front end, developers can work on each part independently, allowing for easier maintenance, scalability, and code reusability. It also enables specialization and collaboration among different teams. This approach helps create more efficient, modular, and user-friendly software applications.

Back and front communicate using data formats which are understandable for both systems, like **JSON**, which is the solution chosen for the specific case.

JSON (JavaScript Object Notation) is a lightweight data interchange format commonly used for transmitting and storing data. It uses a simple syntax to represent objects and arrays, making it easy for humans to read and write. JSON is widely supported and used in web applications and APIs.

The back server must be accessible from two types of clients:

- A web interface (front end) to perform operations on models and datasets created by users.
- Specific games which need to resort to NLP models created in order to understand the sentences of the players.

For this reason, I decided to create the back server with a **REST architecture**.

A **REST server**, short for Representational State Transfer, is a software architectural style used for designing networked applications. It provides a standard set of rules and conventions for communication between client and server systems over the internet. In a REST server, resources are identified by unique URLs, and interactions with these resources are performed using standard HTTP methods like GET, POST, PUT, and DELETE. The server responds with data encoded in formats like JSON or XML. REST servers are widely used for building scalable and interoperable web services and APIs that can be consumed by different client applications.

Summarizing, the AIve architecture is essentially composed of these two main parts:

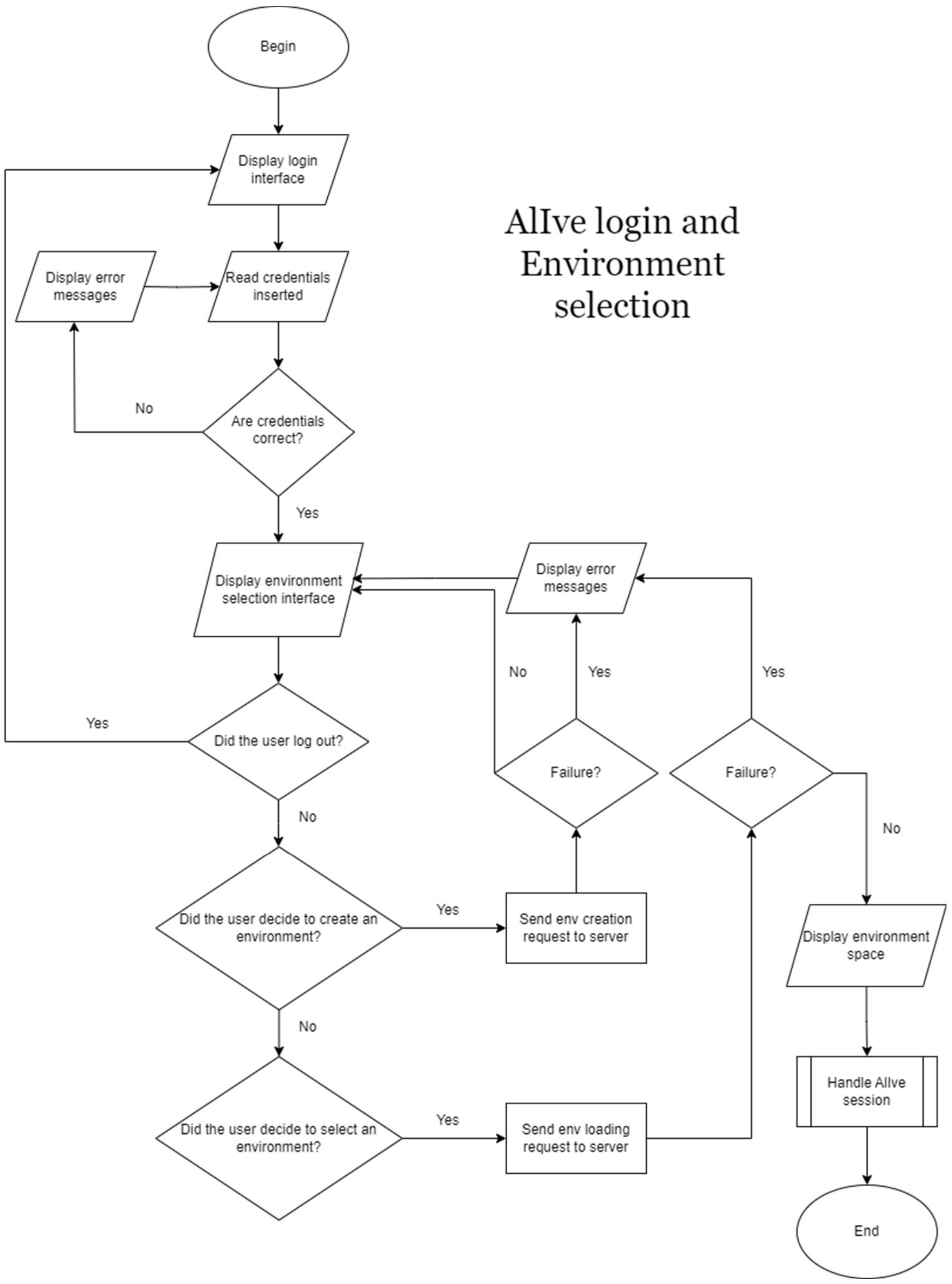
- **REST Server (back)**
    - In order to make the tool compatible with all the engines, the core functionalities should be stored on a REST server, which services can be used through exposed APIs by all programs that need them. All models and datasets owned by users should be accessed only by providing the credentials, so the system guarantees a certain privacy.
      This is beneficial to the game complexity itself because the expensive calculations related with predictions are done by the dedicated server.
- **Web interface (front)**
    - To make this tool easily usable by game developers it should expose a user-friendly interface which can be accessed everywhere, with the only need of a browser.

4.3 – Summarizing the functionalities

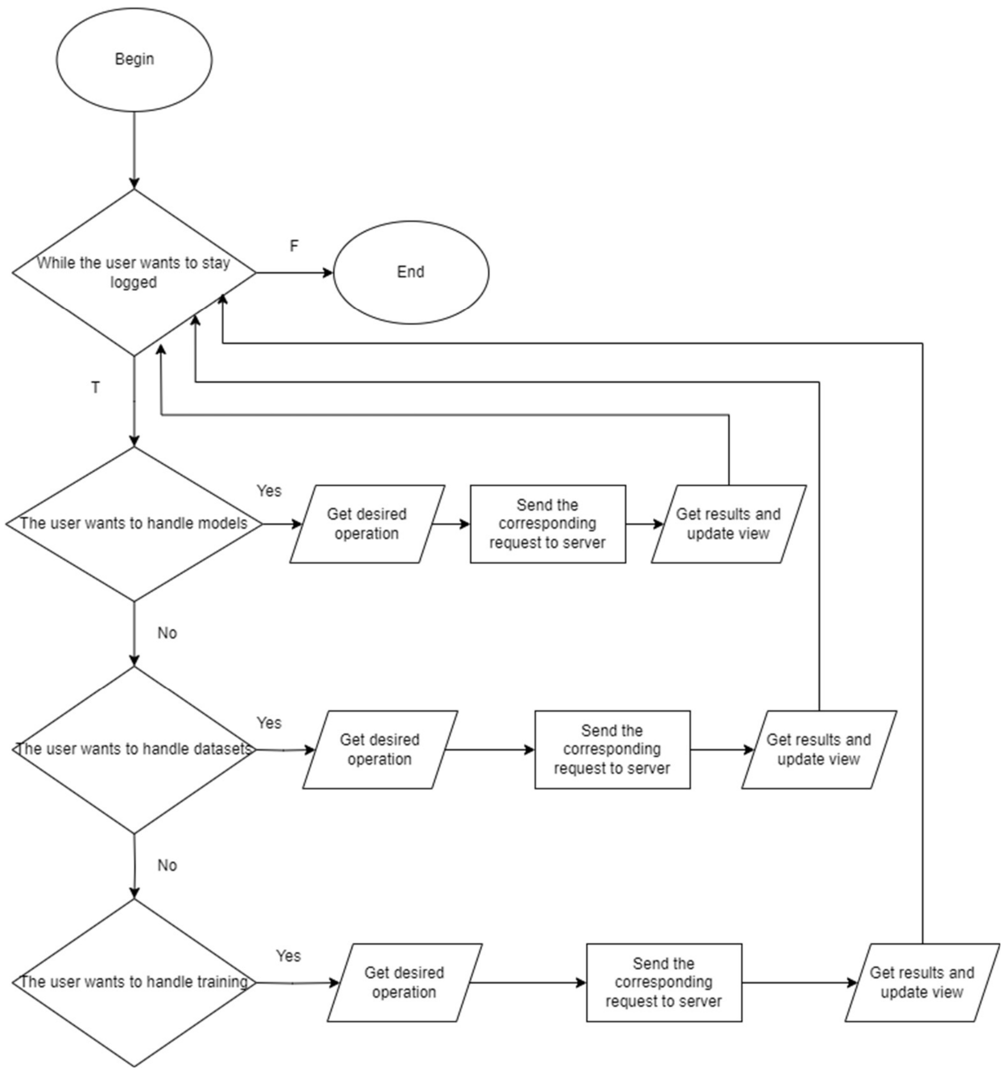
The functionalities I implemented for this tool are the following.

- Models handling: this includes creation of a model, the deletion, the possibility to ask predictions.
- Datasets handling: as for the models, the tool allows the users to create datasets, to import examples stored into csv files specifying the category (train, validation, test), and to delete them.
- Training models: Every model can be trained on a compatible dataset for a certain number of epochs.
- User handling: like the majority of web applications, this tool uses authentication. Only authenticated users can perform the operations mentioned.

The models can be asked for a prediction both through the web interface and through the request of the videogames that need it.


AlIve login and Environment selection
Begin
Display login interface
Read credentials inserted
Display error messages
Are credentials correct?
No
Yes
Display environment selection interface
Did the user log out?
Yes
No
Did the user decide to create an environment?
Yes
Send env creation request to server
Failure?
Yes
Display error messages
No
Did the user decide to select an environment?
Yes
Send env loading request to server
Failure?
Yes
No
Display environment space
Handle AlIve session
End

Handle AIIve Session


Begin
While the user wants to stay logged
F
End
T
The user wants to handle models
Yes
Get desired operation
Send the corresponding request to server
Get results and update view
No
The user wants to handle datasets
Yes
Get desired operation
Send the corresponding request to server
Get results and update view
No
The user wants to handle training
Yes
Get desired operation
Send the corresponding request to server
Get results and update view

4.4 – REST Server

The REST server is responsible for the majority of core functionalities. It allows to create new users, new models, new datasets, and, most importantly, loads the models when predictions are needed, sending the output to the client which made the request.

Every information that needs to be saved permanently, like info about users, models, datasets, training sessions, and others is stored on a relational database handled by MySQL Server.

The language I decided to use to develop the REST server is Python, considering that it’s a flexible interpreted language and nowadays it guarantees access to the majority of Artificial Intelligence libraries.

Specifically, I decided to use Tensorflow and Keras to build customs models, since Tensorflow gives access to Hub, which stores a wide selection of models ready to be finetuned and used.

In addition to that I used various libraries like Pandas to handle csv files, Sickit-learn for utility methods, and Flask to expose the APIs.

The core module of the server stores custom classes to handle models and datasets, specifically we have:

- NLPClassificationModel: abstract class which generically defines the behaviour of the classification models. It implements a few methods, like loading and saving, which shouldn't change between classification models. The other methods are instead implemented by the subclasses. Its main methods are:
    - load_model, save
    - save_train_history_graph: produces and saves a graph showing data related to a training session
    - build, train, predict, evaluate: these methods are implemented by the specific subclasses
- SentenceLevelClassificationModel: This class handles predictions at sentence level; when calling the build method you can specify the output shape, based on the number of classes that you have, and several other parameters, like the optimizer, the learning rate, the base model, and others. The predict function, instead, returns both the class predicted by the model and the probability associated.
- TokenLevelClassificationModel: This class is used to make predictions for every token in the sentence. The build method is similar to the Sentence Level Model, the predict function, instead, returns the class for every token and the entities recognized inside the sentence.
- SentenceLevelClassificationData and TokenLevelClassificationData: These classes take in input csv files storing datasets, respectively for training, validation, and testing, and automatically does the pre-processing of all the data according to the type of model that will use them.

AIlve classifiers are composed of Keras layers disposed in the following way:

- Plain text input layer.
- Pre-processing layer. (Required only if the encoder needs it)
- Encoder layer, which encapsulates a BERT-like model.
- Classification layer, which can be connected to BERT pooled output for sentence level classification or to BERT sequence output for token level classification.

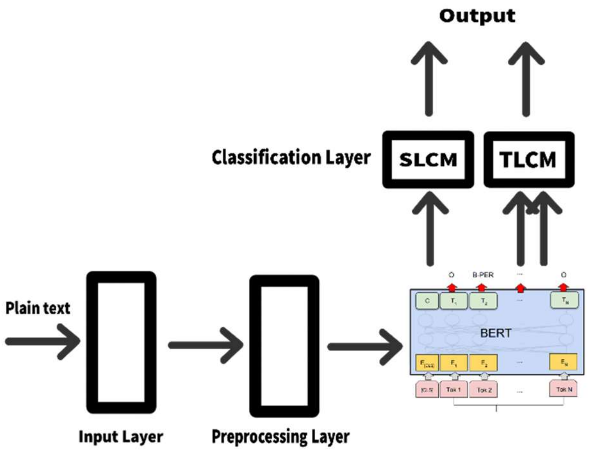


Figure 17 - AIlve models architecture

4.4.1 – Details about pre-processing

The main encoding technique used for this work is **One-hot encoding**, which is performed by the LabelBinarizer class provided by Sickit-learn.

But how does One-hot encoding work?

One-hot encoding is a technique to represent categorical variables numerically. It converts each category into a binary vector where only one element is hot (1) and the rest are cold (0). This encoding allows algorithms to interpret categorical data as continuous values, enabling them to capture relationships between categories. One-hot encoding is commonly used in various tasks, including classification and regression, to process categorical features and enhance the performance of machine learning models.

Sentence Level Classification

When creating objects of type SentenceLevelClassificationData, csv containing train, validation and test examples must be passed.

These csv must have at least two fields, one for the sentence and one for the desired target (Intent, sentiment, …).

When calling the constructor pre-process is automatically performed.

The algorithm:

1. Checks if the csv has the required fields, if not raises an exception
2. Encodes labels for each example using One-hot encoding
3. Saves everything into a new dataframe which will be given in input to a SentenceLevelClassificationModel object when calling the train method.

Note that no pre-process operation is performed on input features, since they will be directly pre-processed from the BERT layer encapsulated into the SentenceLevelClassificationModel's neural network.

Token Level Classification

Again, when creating objects of type TokenLevelClassificationData, csv for train, validation and test examples must be passed.

These csv files in this case must have at least three fields, one for the specific word, one for the sentence index, which specifies to what sentence the current word belongs to, and one for the desired target (NER tag, POS tag, …).

When calling the constructor pre-process is automatically performed.

The algorithm:

1. Checks if the csv has the required fields, if not raises an exception
2. Encodes labels for each example using One-hot encoding
3. Saves everything into a new dataframe which will be given in input to a TokenLevelClassificationModel object when calling the train method.

Even in this case no pre-process operations are performed on input features, since they're directly executed by BERT's pre-processing layer.

4.5 – Web Front

For the web front, instead, I decided to use Vue.js, a JavaScript framework which allows to easily create powerful single page applications.

A Single Page Application (SPA) is a web application that loads once and dynamically updates its content without requiring full page reloads. It achieves a smooth user experience by using JavaScript to handle data retrieval and rendering, providing a responsive and interactive interface within a single web page.

In this way, the web front is way faster, and can still be used like a normal website thanks to the virtual routing, giving the user a friendly browser experience.

The user who accesses to AIlve web application must first login, in order to access to the working space, which is organized through **environments**.

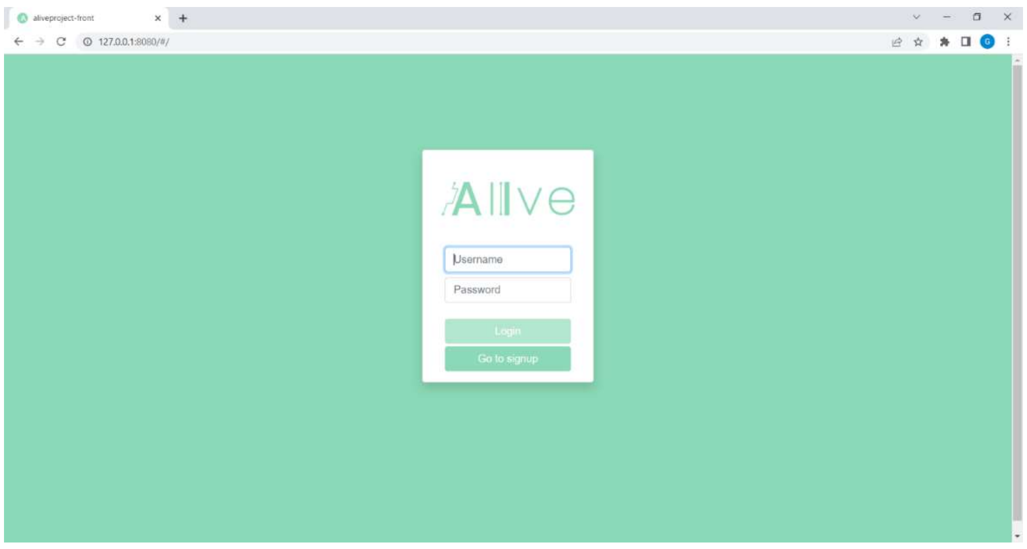


Figure 18 - Alive Login page

Environments are virtual spaces in which the user can define models and datasets.

Once the user has logged in, it will be possible to create or select an environment.

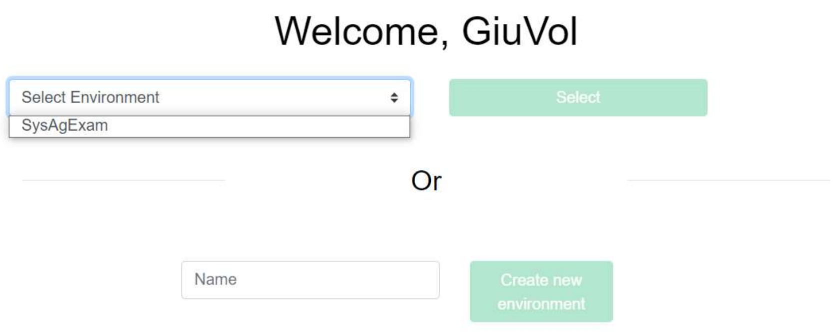


Figure 19 - AIIve Environment Selection

When an environment is selected the user will be able to perform several actions, like creating or deleting models and datasets, check the results of the training sessions, visually represented through graphs, and use the web interface to directly ask some predictions to the desired models.

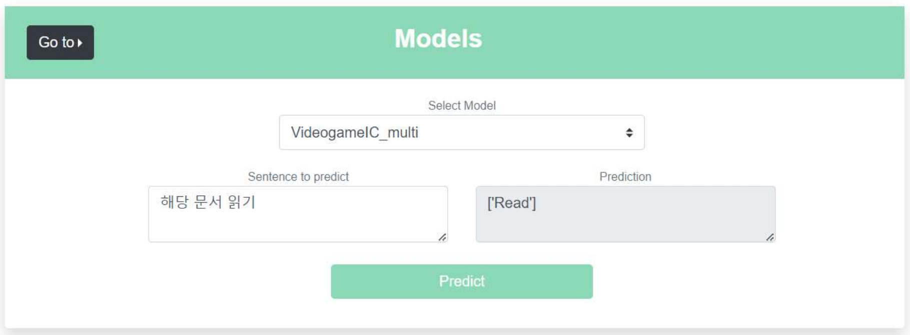
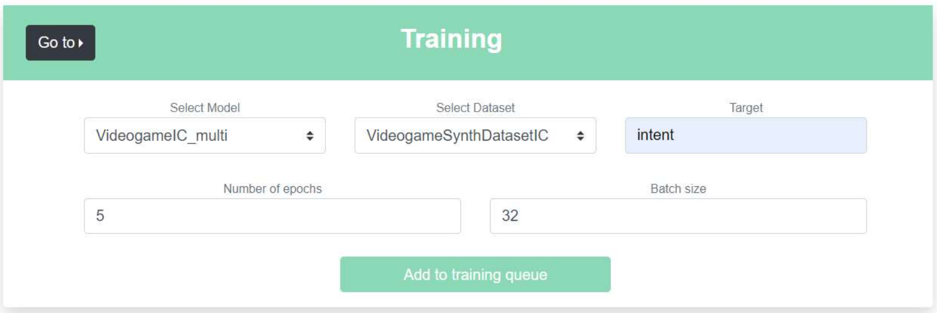


Figure 20 - Asking predictions through Web Interface

To train a model, the user must open the dedicated interface, which allows to specify a model and a compatible dataset, to train for a specific number of epochs with a certain batch size.

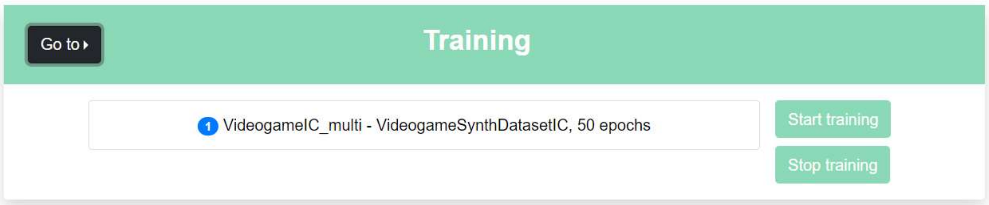


Select Model: VideogameIC_multi
Select Dataset: VideogameSynthDatasetIC
Target: intent
Number of epochs: 5
Batch size: 32

Figure 21 - Adding training session

The training sessions are added to a queue data structure that stores the information about the number of epochs left to complete the training for every session.

The training process can be started from another interface which easily allows to check the queue and the progresses of the current training session.



1 VideogameIC_multi - VideogameSynthDatasetIC, 50 epochs

Figure 22 - Train queue

4.6 – Creating a videogame

Once assured the proper functioning of the NLP tool, I just had to develop a videogame prototype to test the interaction between the game and AIve.

To develop the videogame, I used Unity, which gives access to a wide variety of high-level functionalities useful to handle the graphic part of the game and, most importantly, to a set of functionalities to handle http requests and responses in an efficient way.

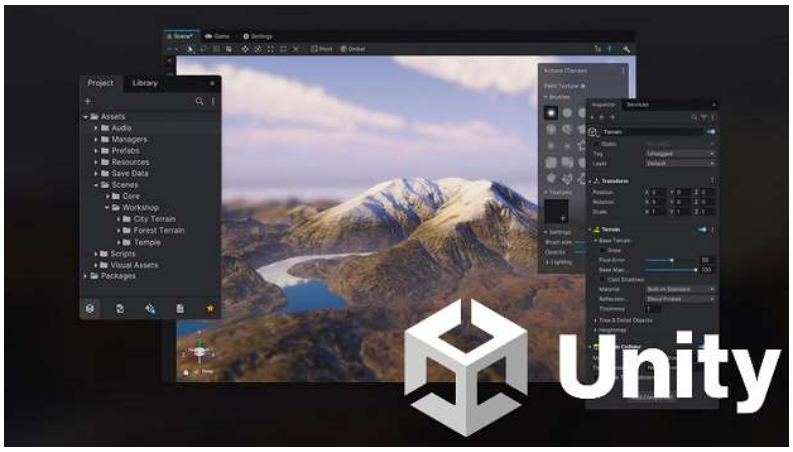


Figure 23 - [https://dotnet.microsoft.com/en-us/apps/games/unity]

The first problem that appears is related to the dataset. How can you create a custom dataset in a simple way? I realized that to do this I could just create a procedural script that uses a GPT model to generate a synthetic dataset, specifically, gpt-3.5-turbo from OpenAI.

The script builds sentences procedurally, by cycling over several parameters, like the type of gamer that is asking a sentence, the type of agent that is dialoguing with the player, the communicated intent, and more…

Once that the sentence is built, the script sends requests to OpenAI, changing the desired temperature, which is the value that determines the creativity of the responses.

The responses of the GPT models correspond to ways to communicate some type of intent.

The script automatically labels these sentences and adds them to the dataset if there are no duplications.

After the generation is done, a task of balancing must be performed.

This procedure is time consuming, but it allowed me to generate a good dataset to create a simple experimental game in which you can give orders to a kid.

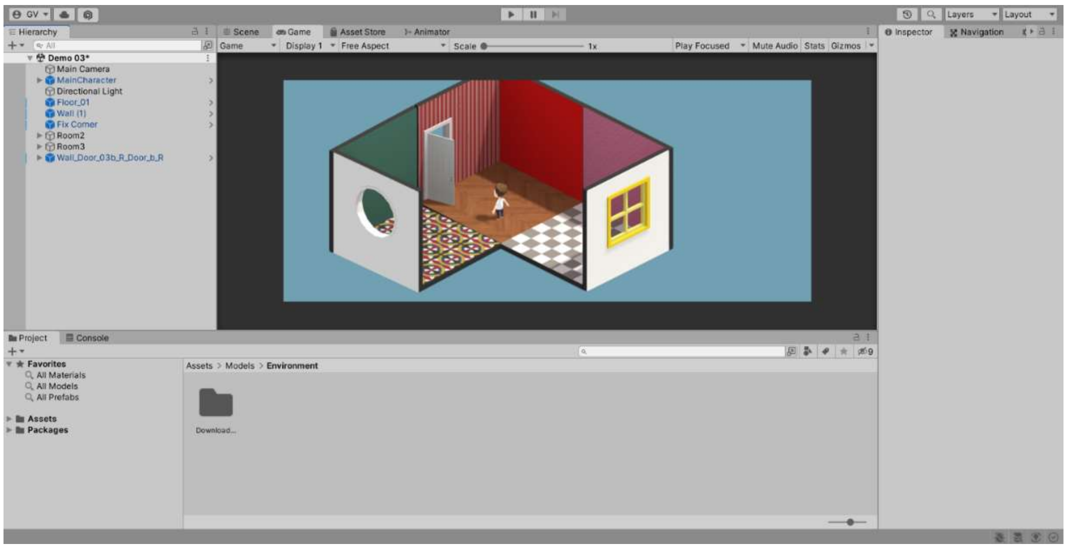


Figure 24 - First demo

4.6.1 – Dataset details

The dataset that I used to train the NLP model for this simple test videogame is focused on the 10 following intents:

- "MoveToRoom"
- "GoDownstairs"
- "GoUpstairs"
- "OpenContainer"
- "PickItem"
- "UseDevice"
- "Read"
- "Write"
- "UseKeyItem"
- "Examine"

It is composed of 18171 examples which are split into train, validation, and test with a proportion of respectively 70:15:15.

The classes in the dataset have been balanced with Sickit-learn utilities.

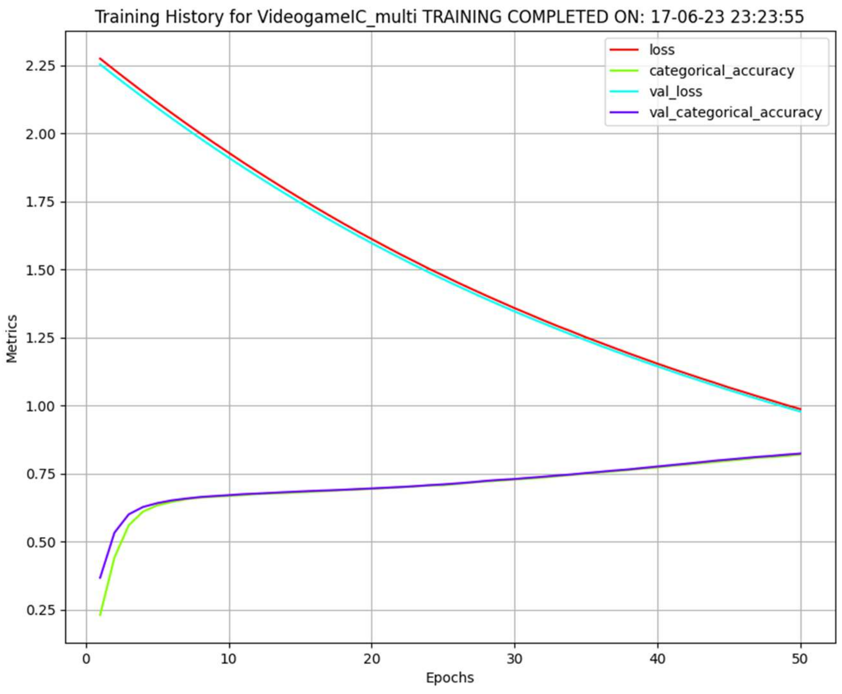
This synthetic dataset has been used to train a Sentence Level Classification Model that uses the "Universal Sentence Encoder Multilingual" encoder, available on TensorflowHub, for the embedding.

This encoder takes in input a variable length text (not pre-processed) and produces a 512-dimensional embedding vector, which encodes the meaning of the sentence. Sentences in different languages, but with the same meaning, will produce similar embeddings.

Universal Sentence Encoder Multilingual supports 16 languages, thanks to this, even though the dataset is composed of examples in Italian, it can make correct predictions on other 15 languages, making the videogame inherently multilingual.

More details on this encoder can be found on the paper "Multilingual Universal Sentence Encoder for Semantic Retrieval" [Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, Ray Kurzweil].

The training of the model produced the following results:



Even though the loss remained higher than the expectations the model works well, giving the correct predictions in several languages.

The main problem with this model is that the encoder used is cased, so the same sentence on different cases could lead to different predictions. I bypassed this problem by simply low-casing the sentences given in input, to guarantee coherent predictions.

In addition to that, it’s hard to generate synthetic multilingual data for NER models, so I gave up on this point due to lack of time, leaving the demo with some missing functionalities, which are the ones that need the entity recognition.

CHAPTER 5- Testing

To test this tool, I had to pass through two main steps:

1. Train some models to check if the NLP part of the project worked.
2. Test the interaction between a developed videogame and the NLP tool to understand if it works well with the rules of a real game.

5.1 – Training models

To check the proper functioning of the NLP tool I trained two models, one for Intent Classification, the other for Named Entity Recognition.

5.1.1 – Intent Classification Model

The Intent Classification Model has been trained on the dataset provided with the work "Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces" [Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, Joseph Dureau].

This dataset focused on seven intents:

- SearchCreativeWork (e.g. Find me the I, Robot television show),
- GetWeather (e.g. Is it windy in Boston, MA right now?),
- BookRestaurant (e.g. I want to book a highly rated restaurant for me and my boyfriend tomorrow night),
- PlayMusic (e.g. Play the last track from Beyoncé off Spotify),
- AddToPlaylist (e.g. Add Diamonds to my roadtrip playlist)
- RateBook (e.g. Give 6 stars to Of Mice and Men)
- SearchScreeningEvent (e.g. Check the showtimes for Wonder Woman in Paris)

Some of the queries of this dataset have been generated for each intent with crowdsourcing methods.

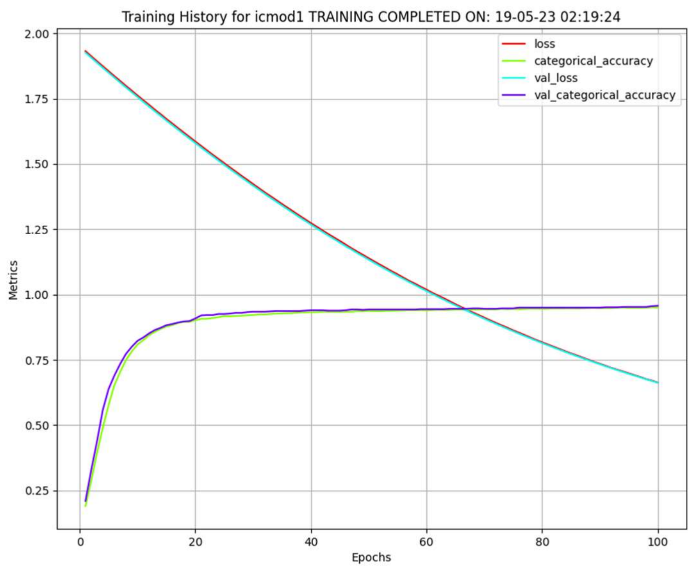
The examples have been distributed in the following way:

- 13084 for training
- 700 for validation
- 700 for testing

This model has been trained for 100 epochs and the training produced the following results:

- Final loss = 0.6122
- Final accuracy = 0.9711

This chart displays the evolution of the metrics during the epochs:



5.1.2 – Named Entity Recognition Model

The Named Entity Recognition Model has been trained on the NER dataset published on Kaggle by Naman Jasawani.

The tags on which this dataset focuses on are 17:

- ORGANIZATION (Georgia-Pacific Corp., WHO, …)
- PERSON (Eddy Bonte, President Obama, …)
- LOCATION (Murray River, Mount Everest, …)
- DATE (June, 2008-06-29, …)
- TIME (two fifty a m, 1:30 p.m., …)
- MONEY (175 million Canadian Dollars, GBP 10.40, …)
- PERCENT (twenty pct, 18.75 %, …)
- FACILITY (Washington Monument, Stonehenge, …)
- GPE (Southeast Asia, Midlothian, …)

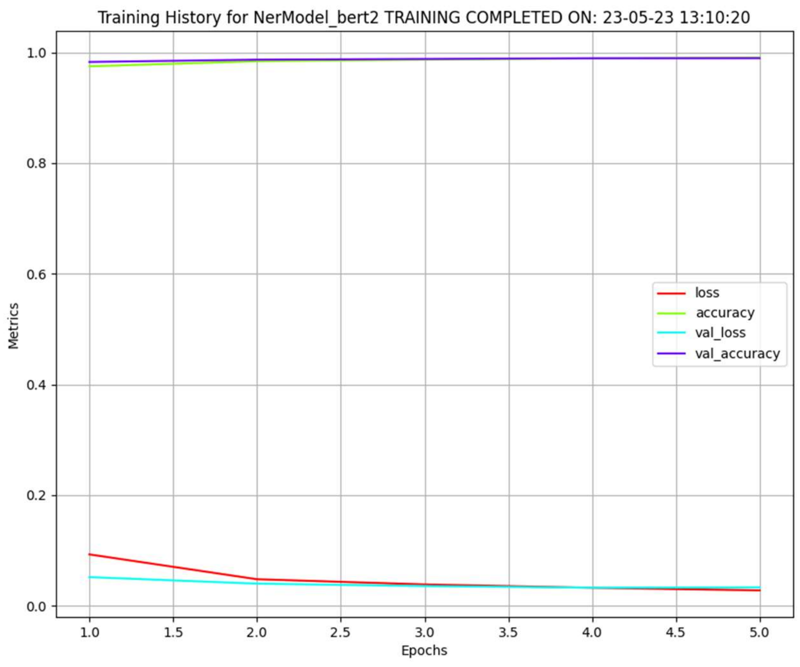
The dataset has 47959 sentences distributed in the following way:

- 33571 for training
- 7194 for validation
- 7194 for testing

This model has been trained for 25 epochs and the training produced the following results:

- Final loss = 0.4254
- Final accuracy = 0.9965

This chart displays the evolution of the metrics during the last 5 epochs:



5.1.3 – Testing a game

The first test videogame developed with AIlve didn’t convince me, since without text generation the interaction between the player and a human character is poor and unnatural.

I decided to develop a new game to then proceed with the test phase, and to do this I collaborated with my friend Riccardo Incampo.

To overcome the absence of text generation we decided to create a game in which you can give commands to a dog in order to achieve some goals, like an environmental puzzle.

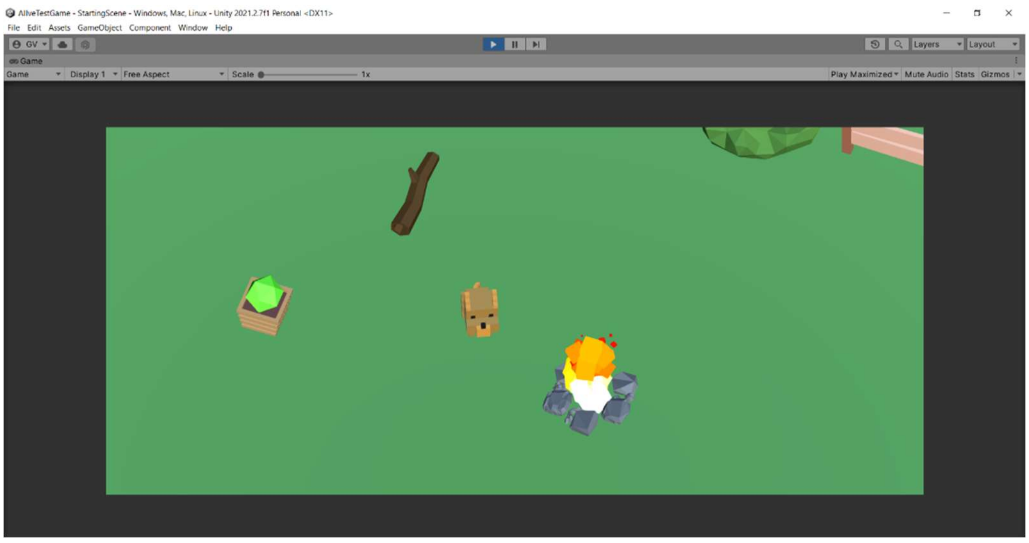

AIlveTestGame - StartingScene - Windows, Mac, Linux - Unity 2021.2.7f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Game
Display 1
Free Aspect
Scale 1x
Play Maximized
Mute Audio
Stats
Gizmos
Layers
Layout

Figure 25 - Game screenshot

I developed this game using free copyright assets downloaded from Unity Asset Store, since it would take too much time to create specific models.

The dog can navigate through the map using NavMesh system from Unity which can find the shortest path between two points in the map using an algorithm similar to the A* algorithm.

I developed the dog AI, which can parse commands by sending requests to AIlve platform.

The actions that the dog can perform are the following:

- Reach a position
- Pick an item
- Leave an item
- Throw an item against something else
- Inspect

The player should give orders to the dog, considering these possible actions. When the dog does not understand the instructions received, a message that states that the dog is confused will appear.

The aim of this demo is to solve a simple puzzle, which consists of finding the keys of the owner.

In order to train the two NLP models useful for this study case (Intent classifier and Named entity recognizer) I created simple procedural datasets, which may not be the best, but have reached good performances.

Intent Classification Model

The dataset for Intent Classification consists of:

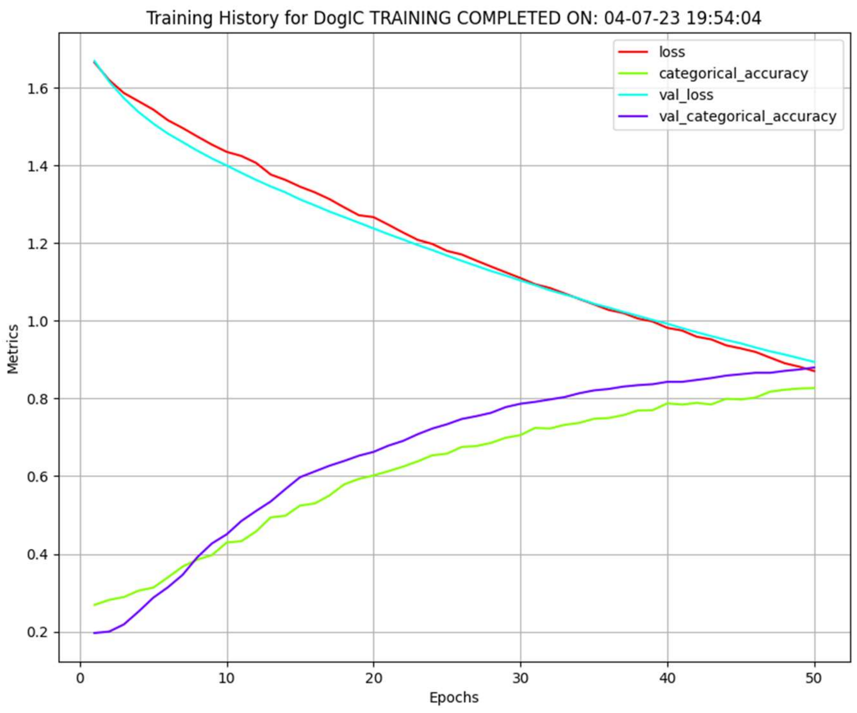
- 2565 examples for train
- 815 examples for validation
- 1800 examples for test

The examples are balanced and the classes correspond to the actions that the dog can perform, mentioned before.

For the preprocess and the encoding before the classification layer, I decided to use again models from the Universal Sentence Encoder family.

Firstly I decided to train the model with the dataset for 50 epochs without finetuning the parameters of the encoder.

This graph shows the evolution of the metrics during the training.



After evaluating the model on test examples I obtained a decent **test accuracy** of **0.8611**.

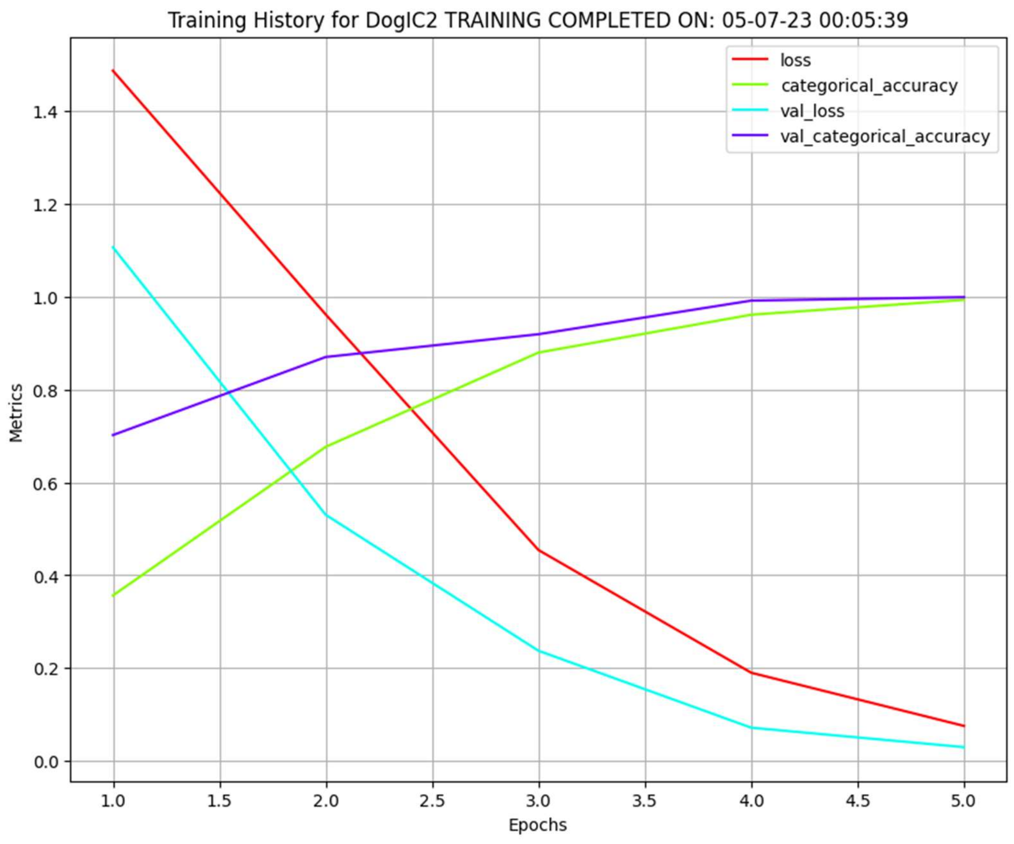
Even though the loss remained high, the performance of this model in the wild went pretty good, correctly recognizing about 9/10 of the intents for the sentences provided.

I decided to train another time a new model, this time for just 5 epochs, but training the parameters of the encoder too.

By doing this, the new model obtained incredible results:

- Train loss: 0.0755
- Train accuracy: 0.9930
- Test loss: 0.0749
- Test accuracy: 0.9927

Here is the graph for the second model.



Of course, I decided to use this model for the intent classification of the game.

Named Entity Recognition Model

This model should recognize the entities involved in an action, for example, the specific item that the dog should pick.

The dataset consists of:

- 10310 words for train
- 4251 words for validation
- 3971 words for test

The tags that the model can recognize, which correspond to the roles that recognized entities can have in a sentence, are the following:

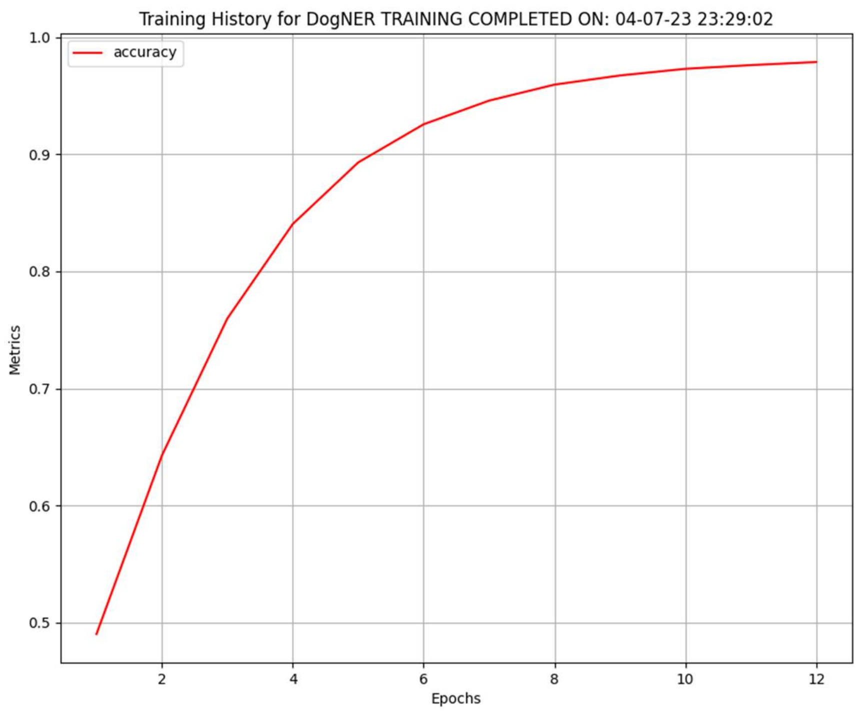
- O (Entities without a role)
- ToReach
- ToPick
- ToLeave
- ToThrow
- ToHit
- To Inspect

These tags are actually split into B(eginning)-tag and I(ntermediate)-tag, which identify the beginning of an entity or the continuing.

For the preprocess and the encoding before the classification layer, I used, even in this case, models from the Universal Sentence Encoder family.

For the first try I trained the NER model for 12 epochs and, even though the accuracy has grown well, the model didn't work properly in the wild, giving the O tag to almost every token of every sentence.

This graph shows how the accuracy evolved through the epochs:

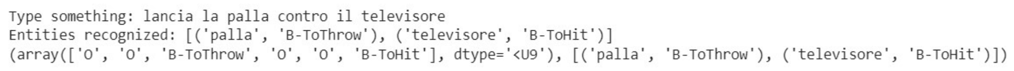


To achieve better performances in the wild, I trained a new NER model, this time training the encoder parameters too.

The results obtained with this trick are the following.

- Train loss: 0.0014
- Train accuracy: 0.9997
- Test loss: 0.0011
- Test accuracy: 0.9991

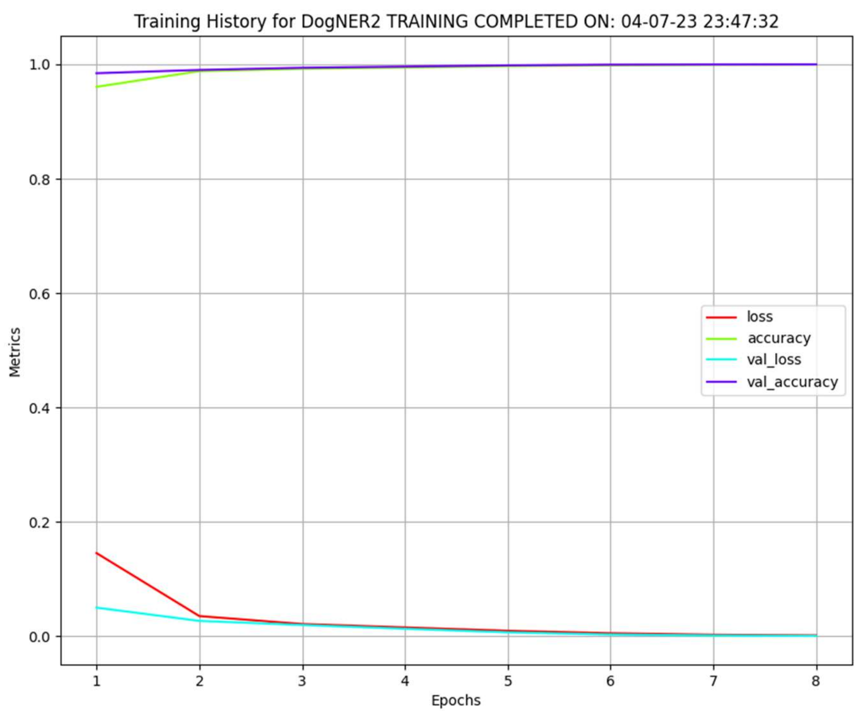
In addition to that, the new model worked really well in the wild, recognizing properly the entities in the sentences given.



```
Type something: lancia la palla contro il televisore
Entities recognized: [('palla', 'B-ToThrow'), ('televisore', 'B-ToHit')]
(array(['O', 'O', 'B-ToThrow', 'O', 'O', 'B-ToHit'], dtype='<U9'), [('palla', 'B-ToThrow'), ('televisore', 'B-ToHit')])
```

Figure 26 - New NER model prediction

The following graph shows the evolution of the metrics during the training:



CHAPTER 6 – Conclusions and future developments

With this project I wanted to highlight how it could be possible to develop a simple workflow that aims to use Artificial Intelligence in the field of videogame development without breaking the rules that characterize the world game, and I think that the obtained results are encouraging.

In the future I will try to keep this project on, by improving it and fixing all the problems that by now are overwhelming, due to lack of time and computational resources. This project, in fact, requires a high computational power, and I often struggled to pre-process data and train models just using my computer.

The next important step would be to load the project on a dedicated server, so that it will have the resources needed to train models and to respond to the clients' requests.

I'm currently working to implement a new synthetic dataset generation algorithm to be independent of OpenAI APIs.

In addition to that, the tool can be enlarged by adding new types of models like regressors, text generation models and even models that are not related to Natural Language Processing, like computer vision technologies for instance.

In conclusion, I just hope that with this project the world games of the future will be more AIIve than ever.

BIBLIOGRAPHY

1. “Introduction to Game Development” [Steve Rubin, 2010]
2. "Multilingual Persuasion Detection: Video Games as an Invaluable Data Source for NLP" [Teemu Pöyhönen, Mika Hämäläinen e Khalid Alnajjar, Helsinki University, 10/07/2022]
3. "Natural Language Processing in Game Studies Research: An Overview" [José P. Zagal, Noriko Tomuro, Andriy Shepitsen]
4. "The Wisdom of the Gaming Crowd" [Robert Jeffrey, Pengze Bian, Fan Ji, Penny Sweetser]
5. "Generative Agents: Interactive Simulacra of Human Behavior" [Joon Sung Park, Joseph C. O’Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, Michael S. Bernstein, 07/04/2023]
6. "Generating Role-Playing Game Quest Descriptions With the GPT-2 Language Model" [Susanna Värtinen, Aalto University, 10/12/2021]
7. “Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces” [Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, Joseph Dureau]
8. "Attention is All You Need" [Vaswani et al., 2017]
9. “Multilingual Universal Sentence Encoder for Semantic Retrieval” [Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, Ray Kurzweil]